Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di laurea in Ingegneria dell'Informazione

Sviluppo di un sistema distribuito di misura per la qualità delle acque

Relatore
Prof. Alessandro Pozzebon

Laureando
Matteo De Facci

Anno accademico 2021-2022

Indice

Introduzione

L'acqua è un bene fondamentale e ancor più oggi il suo controllo è un problema centrale.

Il monitoraggio della qualità dell'acqua è quindi un aspetto fondamentale per l'uomo e per l'intero ecosistema. Monitorare in punti specifici come fiumi, torrenti, laghi, pozzi e falde è un compito oneroso, dispendioso e spesso difficilmente praticabile.

Lo scopo del progetto su cui si basa il seguente elaborato è sviluppare un sistema di controllo remoto di alcuni parametri delle acque; nello specifico: *pressione* e *temperatura*. Il dispositivo viene sviluppato con l'obiettivo di permettere una facile integrazione con sensori supplementari per una raccolta dati ancora più completa.
La scelta delle grandezze misurate è basata sull'interesse dello studio del livello, inteso come profondità, dell'acqua.

Il progetto quindi comprende la parte di raccolta dati attraverso i sensori, la tecnologia di trasmissione basata su protocollo *LoRaWAN* e lo sviluppo della componentistica e del software per assicurare un'autonomia adeguata al tipo di dispositivo.
L'utenza finale avrà a disposizione un server da cui accedere ai dati raccolti e il controllo dei dispositivi.

I vantaggi principali di un prodotto *self-made* di questo tipo consistono nella versatilità e nel basso costo del dispositivo fisico che ne permettono una vasta distribuzione sul territorio e di conseguenza un controllo capillare delle acque.

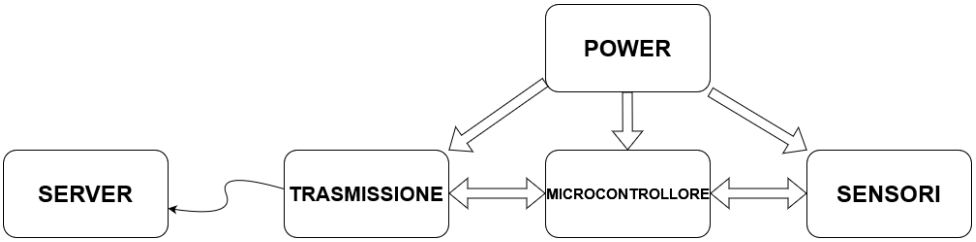


Figura 1: Flowchart del progetto.

Capitolo 1

Sensore di pressione

1.1 Pressione dell'acqua

Un solido immerso in acqua viene sottoposto ad una forza su tutta la superficie di contatto. A sua volta, per il principio di azione e reazione, l'acqua subisce la stessa forza ma in direzione opposta. Se il solido e l'acqua sono fermi l'uno rispetto all'altra, le forze (**F**) di contatto sono dirette perpendicolarmente a tutti gli elementi di superficie (**S**) di contatto. La pressione si definisce come il rapporto tra queste due grandezze: $P = \frac{F}{S}$.

A livello dimensionale dato che la pressione è una forza per unità di superficie la sua unità di misura fondamentale è $\left[\frac{N}{m^2}\right]$ che viene denominata **Pascal** [**Pa**]. La pressione però ha altre unità di misura non appartenenti al sistema internazionale, usate in diversi contesti scientifici. L'interesse in questo contesto è legato alle conversioni tra le unità più utilizzate, e vengono riportate nella tabella 1.1.

	Equivalenza in Pascal
1 bar	100.000 Pa
1 mbar	100 Pa
1 atm	101.325 Pa

Tabella 1.1: Comparazione con altre unità di pressione.

La pressione in un qualsiasi punto all'interno di un fluido è la stessa in tutte le direzioni ed aumenta con l'aumentare della profondità. Si può dimostrare che, preso un elemento si superficie **S** ad una certa altezza **h** dalla superficie, su di esso agisce una forza dovuta al peso **Ps** della colonna d'acqua sovrastante con volume $V = S \cdot h$. Il peso `Ps` è il prodotto del volume **V** per il peso specifico λ. Pertanto la pressione ad una profondità **h** risulta:

$$P = \frac{Ps}{S} = \frac{V \cdot \lambda}{S} = \frac{S \cdot h \cdot \lambda}{S} = h \cdot \lambda \tag{1.1}$$

Questa è la **Legge di Stevino**.

Il peso specifico di un materiale è $\lambda = \rho \cdot g$ dove ρ e g sono rispettivamente la *densistà*

del fluido e *l'accelerazione di gravità*. Nel caso dell'acqua sono entrambe grandezze note:

$$\rho = 997kg/m^3, \qquad g = 9.81m/s^2$$

Il livello di acqua di un recipiente può essere quindi calcolato dalla legge di Stevino 1.1 conoscendo la pressione **P** sul fondo dello stesso.

$$h = \frac{P}{\rho \cdot g} \quad \left[\frac{N/m^2}{kg/m^3 \cdot m/s^2}\right] = [m] \tag{1.2}$$

1.2 Sensore di pressione MS5803

La scelta del sensore di pressione è stata effettuata in base ai seguenti parametri di ricerca:

- Disponibilità sul mercato per ovvi motivi di reperibilità.
- Isolamento per utilizzo in immersione.
- Portata e sensibilità: in base alle esigenze dei ricercatori che hanno richiesto il prototipo. Nello specifico: 3 m di prodondità che corrispondono a circa 0.3 bar e una sensibilità di *mezzo centimetro* cioè 0.5 mbar [1].
- Protocollo di comunicazione con il microcontrollore. Scelto in base ai protocolli disponibili nel processore scelto (per maggiori informazioni si rimanda al capitolo 3).

Per gli scopi del progetto si è scelto il sensore ***MS5803*** della casa produttrice *TE Connectivity*.

Figura 1.1: Sensore di pressione e temperatura MS5803.

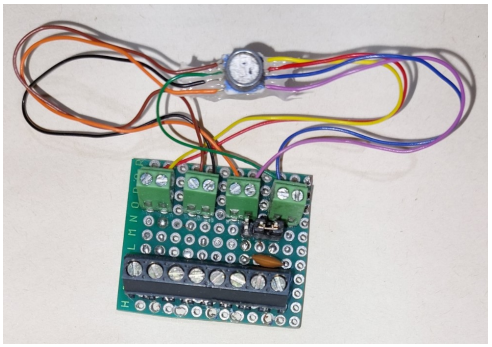
Questo dispositivo è basato su tecnologia *MEMS (Micro Elettro-Mechanical System)* cioè dispositivi integrati sullo stesso substrato di semiconduttore che uniscono la tecnologia dei sensori e attuatori (meccanici, chimici, elettrici, etc.) con l'elettronica per la gestione dei processi. L'integrato ***MS5803*** ha al suo interno un ADC a 24bit ed un microcontrollore che fornisce in *output* il dato nei protocolli standard I^2C e *SPI*. Per

[1] Presi due valori di altezza della colonna di fluido $h1, h2$ la differenza di pressione tra questi punti risulta $\Delta P = \rho \cdot g \cdot h2 - \rho \cdot g \cdot h1 = \rho \cdot g \cdot (h2 - h1) = \rho \cdot g \cdot \Delta h$. Valutando la formula per una differenza di altezza di $0.5 \cdot 10^{-2}$ m si ottiene il valore precedentemente riportato.

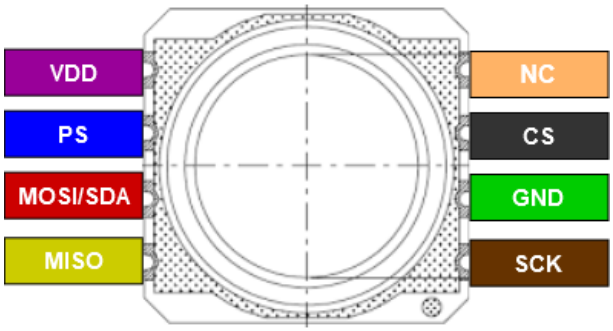
quanto riguarda le caratteristiche di misura il dispositivo ha una portata di 14 bar e una risoluzione di 0.2 mbar [2], valori conformi al prodotto cercato.

1.3 Elettronica di supporto

A livello circuitale il sensore **non** richiede componenti esterni oltre ad un condensatore da 100 nF tra il PIN di alimentazione e *GND*. Lo scopo di questo condensatore è di stabilizzare la tensione durante le fasi di conversione e quindi fornire la massima precisione.



(a) Circuito per il sensore MS5803 su millefori.



(b) Pin-out MS5803.

Nel circuito realizzato su scheda millefori vi è stato inoltre aggiunto un selettore per il pin *PS (Protocol Select)* intercambiabile tra *GND* e *Vcc* per selezionare uno tra i protocolli I^2C e *SPI*.

Nel progetto viene utilizzato il protocollo *SPI*; il circuito realizzato viene mostrato in figura 1.3. Questa scelta è dovuta alla necessità di ridurre lo spazio occupato dal software nella memoria *FLASH* del microcontrollore; dato che il modulo di trasmissione (descritto in seguito nel capitolo 2.2.2) comunica in SPI, l'utilizzo di ulteriori librerie avrebbe reso insufficiente la memoria.

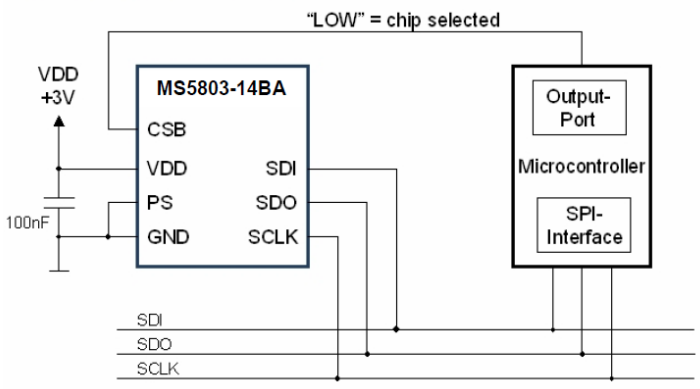


Figura 1.3: Circuito per la comunicazione SPI

[2]Dati forniti nel *datasheet* del dispositivo.

Capitolo 2

Protocollo di comunicazione

In questo capitolo viene affrontato il problema della trasmissione dei dati ad un server per permettere agli utenti di accedervi in remoto. Verranno valutati una serie di protocolli potenzialmente validi per poi scegliere il migliore per lo scopo.

È fondamentale che il protocollo sia affidabile ma soprattutto richieda poca energia per funzionare dato che il prodotto finale ha nell'autonomia il nodo principale.

2.1 Protocolli candidati

In questa sezione vengono descritte le caratteristiche principali delle tecnologie prese in considerazione per il progetto. Le informazioni riportate sono relative agli ambiti di interesse allo scopo, non viene data una descrizione completa dei protocolli.

1. **GSM**. Questo è il protocollo usato nelle precedenti versioni della rete cellulare; anche conosciuto come *2G*. Permette l'accesso alla rete tramite un identificativo che viene fornito dalla scheda SIM. Richiede quindi, oltre al modulo di comunicazione, l'apertura di un piano di traffico dati. Per la copertura si basa sui ripetitori per dispositivi mobili, estenderla richiede costi notevoli. Il vantaggio principale si trova nella quantità di dati che si possono trasmettere; di contro i consumi elevati non la rendono la tecnologia migliore per dispositivi *IoT low power*.

2. **NB-IoT**. Grazie a questa tecnologia a bassa potenza è possibile connettere un ampio numero di dispositivi IoT utilizzando le reti mobili esistenti. La tecnologia è caratterizzata dal basso consumo e dalla gestione ottimale della banda utilizzata, NB-IoT offre la possibilità di trasmettere dati bidirezionali in modo efficiente, affidabile e a lunghe distanze. Lo svantaggio di questa tecnologia è nel costo di implementazione: necessita di un hardware complesso (quindi moduli costosi) e di una SIM - *LTE*.

3. **SIGFOX**. SigFox offre una connettività, similmente alla rete cellulare, che si adatta ad applicazioni fisse e mobili: una rete a lunga portata e a basse velocità, che permette la comunicazione di pochi dati tra gli apparecchi connessi, senza passare da modem 2G, 3G o 4G o da infrastrutture wireless complesse. In questo modo si

riducono significativamente i costi e il consumo di energia delle periferiche collegate. La connessione a bassa velocità tra gli oggetti collegati è possibile grazie all'impiego della tecnologia radio Ultra Narrow Band (UNB) ovvero 'banda ultra stretta' con frequenza di base libera da licenza. Sulla rete SigFox gli oggetti possono trasmettere fino a 140 messaggi ogni giorno da 12 byte ciascuno. La rete è bidirezionale ovvero gli oggetti trasmettono ma possono anche ricevere dati attraverso una piattaforma cloud; i downlink sono limitati a 4 messaggi al giorno. La copertura della rete *SigFox* è molto estesa in Europa e anche in Italia (dove il servizio è fornito in maniera monopolistica da *NetTrotter*); il servizio è però fornito a pagamento periodico per ogni terminale connesso alla rete. L'utente può solamente progettare il device/sensore e non potrà installare alcun tipo di gateway per estendere la copertura.

4. **LoRaWAN**. Questa è una tecnologia molto simile a *SigFox*: basso consumo e basse velocità di trasmissione. Le differenze principali si riscontrano nella quantità di messaggi che possono essere inviati: in LoRaWAN non c'è un limite giornaliero (esiste comunque un regolamento che limita l'occupazione della banda per singolo dispositivo). La seconda differenza è legata all'estensione sul territorio, inferiore rispetto a SigFox. *LoRa Alliance*, che gestisce LoRa, è un'associazione aperta e fornisce il servizio in maniera gratuita. Gli utenti stessi possono installare gateway nel territorio e collegarli alla rete per estendere la copertura di LoRaWAN.

5. **Altri**. *Bluetooth Low Energy* e *Wi-Fi*: entrambe tecnologie molto diffuse con l'evidente vantaggio in termini di velocità di trasmissione. Esistono versioni in cui i consumi sono molto ridotti con lo svantaggio di avere una portata molto limitata (meno di 100 m). Queste soluzioni sono state scartate per incompatibilità con gli utilizzi del progetto.

I fattori principali della valutazione sono stati il costo e la copertura. Velocità di trasmissione elevate non sono generalmente richieste per dispositivi, come quello in oggetto, di raccolta dati. La libertà nei messaggi e l'economicità del servizio ha fatto propendere la scelta per la tecnologia **LoRaWAN**.

2.2 LoRaWAN

LoRaWAN è di base una **LPWAN** (*Low Power Wide Area Network*) cioè una rete di telecomunicazione wireless progettata per consentire comunicazioni a lungo raggio e basse velocità di trasmissione *(bit rate)* ma con un ridotto consumo di potenza.

LoRaWAN nasce da *LoRa (Long Range)*, un formato di modulazione unico detto "**Chirp Spread Spectrum**" (CSS) e rappresenta unicamente il **livello fisico** della comunicazione. Diverse tecnologie LPWAN utilizzano frequenze con o senza licenza e specifiche proprietarie o aperte. In questo caso *LoRa* in Europa sfutta la banda libera *ISM*

(Industrial, Scientific, and Medical) a *868MHz* [1].

Al livello fisico *LoRa* è stato in seguito aggiunto un livello *MAC* denominato *LoRaWAN* (***Wide Area Network***): il quale estende il primo ad Internet. La specifica *LoRaWAN* è un livello software che definisce come i terminali devono sfruttare la modulazione *LoRa*, per esempio quando trasmettono e ricevono messaggi.

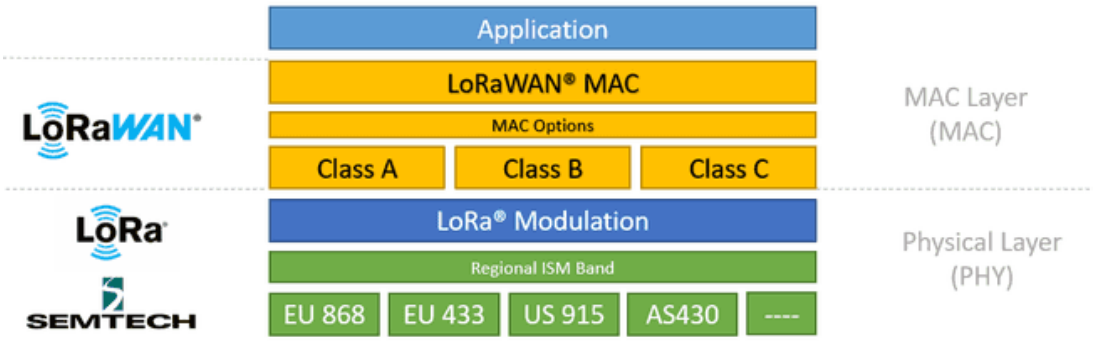


Figura 2.1: Livelli di rete LoRa e LoRaWAN.

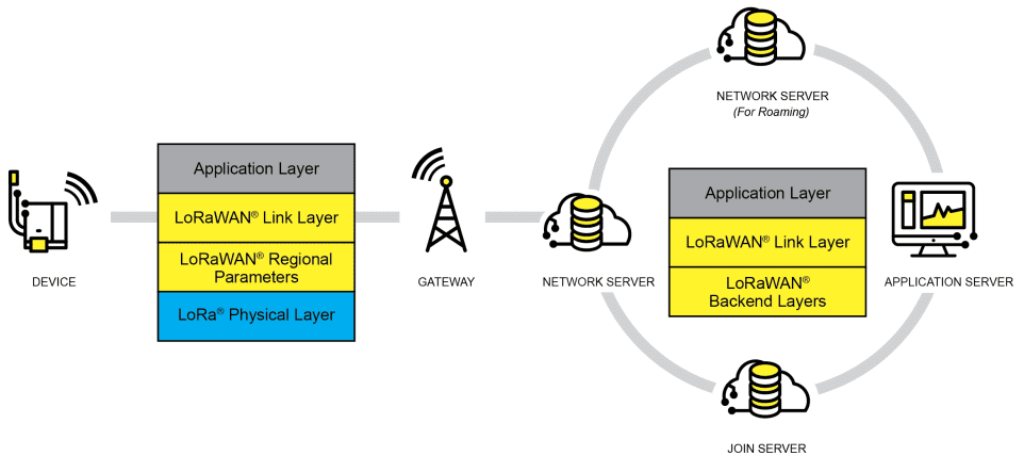


Figura 2.2: Architettura di rete *LoRaWAN*.

Le componenti essenziali che compongono l'architettura *LoRaWAN*, come riportato in figura 2.2, sono: gli **end-device** (principalmente sensori), i **gateway** ed infine i **server di rete**. I gateway sono connessi al server di rete centrale attraverso Internet e gestiscono la trasmissione dati con gli end-device mediante *LoRa*. Il *network server* controlla il livello MAC della rete LoRaWAN e fornisce l'integrazione con altre piattaforme per creare l'applicazione di gestione dei dispositivi. Gli end-device non sono esclusivamente legati ad un singolo gateway, ma possono inviare le informazioni a tutti i gateway entro una certa area. I server di rete e i gateway possono essere pubblici o privati: questo ha facilitato la

[1] La modulazione*LoRa*™ è stata inventata nel 2010 dalla startup francese Cycleo; successivamente è stata acquisita da Semtech nel 2012.

diffusione della tecnologia perchè chiunque può inserire nella rete (senza obblighi a priori) gateway, permettendone anche ad altri utenti l'uso.

2.2.1 Classi A,B,C

La comunicazione in reti *LoRaWAN* è nativamente **bidirezionale** ma gli eventi di *uplink* (da end-device a server) sono predominanti. LoRaWAN ha tre classi per i terminali: A,B,C. Queste si differenziano per consumi e fasi di trasmissione; di seguito verrà approfondita solamente la classe A essendo quella utilizzata nel progetto.

La *classe A* è la classe di default che tutti i dispositivi che adottano il protocollo LoRaWAN devono implementare. Un device può inviare un *uplink* in qualsiasi istante e

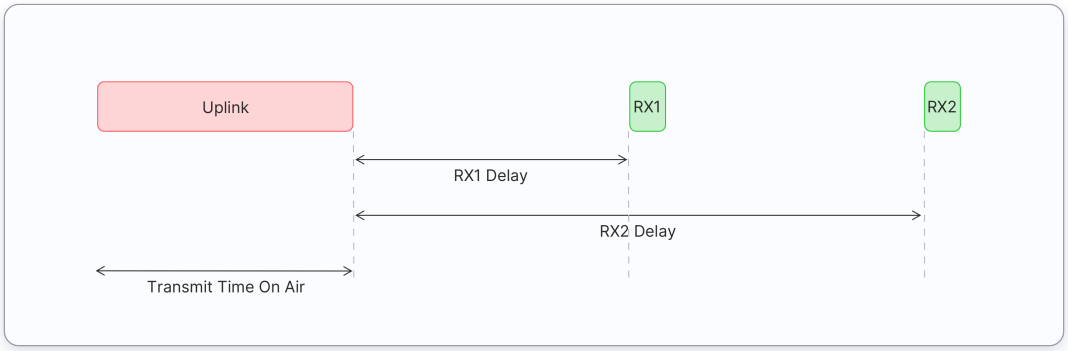


Figura 2.3: Fasi di trasmissione *classe-A*

alla conclusione di questo il dispositivo apre due finestre di ricezione: **RX1, RX2**. Se viene ricevuto un messaggio in una di queste due fasi (mutuamente esclusive) allora inzia la fase di elaborazione del contenuto. Se invece non era previsto o per ritardi non arrivano *downlink* il server dovrà aspettare le prossime finestre di ricezione per inviare i dati, quindi al successivo uplink.

A differenza della classe A, la classe B ha una minore latenza in downlink perchè periodicamente vengono spediti dei *beacon*: messaggi da gateway a end-device per mantenere la sincronizzazione temporale tra i due.

Infine la classe C è sulla falsariga della prima classe, con la differenza che la seconda finestra di ricezione viene mantenuta aperta fino al sucessivo uplink. Chiaramente queste ultime due classi per mantenere il collegamento attivo (tra gateway e dispositivo) consumano molta più energia. Per questo motivo sono poco comuni in applicazioni a ridotti consumi.

2.2.2 Modulo RFM96

L'*RFM96* è un integrato ideato per comunicazioni radio a varie frequenze e permette diverse modulazioni di canale. È caratterizzato da un modulo trasmettitore basato su chip *SX1276* di *Semtech* che fornisce una comunicazione a spettro espanso a lungo raggio e alta immunità alle interferenze pur riducendo al minimo il consumo di corrente. Grazie alla

tecnica di modulazione *LoRa™* lo rende ottimale per qualsiasi applicazione che richiede range o robustezza. In Europa per LoRaWAN è in uso comunemente la banda *ISM 863–870 MHz*. Questa banda è regolamentata in linea generale da normative Europee che ne limitano la potenza e il tempo di trasmissione (duty cycle[2]).

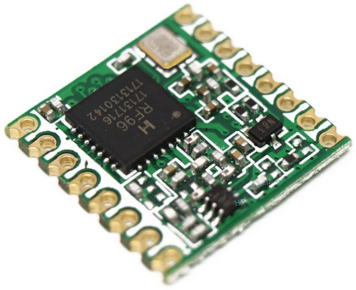


Figura 2.4: Modulo RFM96

L'*RFM96* permette di interfacciarsi con un microcontrollore esterno tramite protocollo *SPI*; inoltre sono presenti pin digitali per la gestione degli eventi. Lo schema riassuntivo dei collegamenti viene riportato nella tabella 2.1.

Tipo	**PIN rfm96**	**PIN Atmega328p**	**Descrizione**
Alimentazione	VCC GND	3.3V GND	Alimentato da batteria Massa comune
SPI	SCK MISO MOSI NSS	SCK, pin 13 MISO, pin 12 MOSI, pin 11 pin 9	Segnale di clock per sincronizzazione Master Input Slave Output Master Output Slave Input Chip select
Digitali	RST DIO0 DIO1	pin 14 pin 15 pin 16	PIN di reset rfm96 Conclusione fasi TX-RX Conclusione finestra di ricezione

Tabella 2.1: Collegamenti RFM96

2.3 The Things Network

The Things Network, in breve **TTN**, è un ecosistema internazionale per l'**IoT** che comprende reti, dispositivi e soluzioni che utilizzano la tecnologia LoRaWAN. The Things Network implementa The Things Stack, una rete LoRaWAN *open-source*. TTS fornisce server di rete in cui è possibile gestire le reti LoRaWAN private e pubbliche. Per il progetto si è fatto uso di questa rete per la sua diffusione e la facilità di utilizzo. Per poter entrare nell'ecosistema è necessario iscrivere l'*end-device* al *network TTN* e ricevere

[2]Il ciclo di servizio (duty cycle) – è definito come il massimo tempo di trasmissione per ogni ora di servizio.

le credenziali di accesso che, durante l'esecuzione, il terminale utilizzerà per connettersi e iniziare la trasmissione.

TTN mette a disposizione una sezione in cui è possibile elaborare il formato dei messaggi che arrivano al server detti **payload**. La formattazione *payload* è un codice che preprocessa i dati in arrivo dal device prima di renderli disponibili all'utente o per l'esecuzione di determinate azioni.

I dati di payload del progetto sono le letture di pressione e temperatura da parte del sensore; di conseguenza sono stati creati i campi "*temperature*" e "*pressure*" e conterranno in array le misure nei diversi istanti temporali. Ulteriori campi, aggiunti per il controllo di tempistiche e numero di acquisizioni, sono "*timeacq*" e "*nsamples*". Questi ultimi due non vengono trasmessi dal terminale ma sono variabili definite per integrare al meglio la dashboard del progetto. In figura 2.5 è riportato un esempio dei campi dati in seguito all'elaborazione del payload.

Per completezza viene riportato il codice in Javascript del payload nell'appendice A.

2.4 Dashboard

Per la visualizzazione finale dei dati raccolti viene fatto uso della dashboard fornita da ***Datacake***. Questo sito permette la creazione di pagine in cui è possibile inserire dei *widget* come grafici, pulsanti, slider etc. Per la visualizzazione di temperatura e pressione sono stati inseriti due grafici in funzione del tempo, impostandone assi e stile.

Datacake, come TTN, mette a disposizione una sezione in cui inserire la formattazione del payload. Questa volta i messaggi in ingresso arrivano da TTN e non direttamente dal dispositivo, perciò sono già in formato personalizzato come descritto nel precedente paragrafo. Il codice inserito nella sezione *Payload decoder* di Datacake crea un array di misure assegnando ai rispettivi campi i valori contenuti negli array inviati da TTN, con l'aggiunta dell'istante di tempo in cui la singola misura è stata raccolta. Conoscendo l'intervallo di tempo di acquisizione dati da parte del sensore (viene riportato anche in TTN nel *field* "*timeacq*") e il tempo in cui i dati sono stati ricevuti, si ricostruiscono gli istanti temporali in cui le misure sono state raccolte. In seguito l'array così creato verrà interpretato da Datacake inserendo i valori delle misure e i rispettivi istanti di tempo nei due grafici della dashboard. In figura 2.6 viene riportato un esempio dell'assegnazione di valori multipli ai rispettivi campi in Datacake. Ad ogni misura viene associato un istante di tempo *timestamp* (quello in cui è stata acquisita) in formato **UNIX time stamp**[3].

Si rimanda all'appendice B per il codice di questo decoder.

[3] *UNIX time stamp* è un sistema di misurazione del tempo in cui si considerano solo il numero di secondi trascorsi dalla mezzanotte del 1° Gennaio 1970. Il formato prende il nome dal sistema operativo **UNIX** che lo adotta.

```
"decoded_payload": {
  "nsamples": 4,
  "pressure": [
    975.97,
    975.53,
    975.72,
    975.7
  ],
  "temperature": [
    30.48,
    30.42,
    30.49,
    30.58
  ],
  "timeacq": 0.5
},
```


Figura 2.5: Esempio campi dati TTN, post elaborazione dati inviati dal sensore.

```
TIMEACQ = 0.5 (timestamp: auto)
NSAMPLES = 4 (timestamp: auto)
PRESSURE = 975.97 (timestamp: 1661701622 -> 2022-08-28T15:47:02)
TEMPERATURE = 30.48 (timestamp: 1661701622 -> 2022-08-28T15:47:02)
PRESSURE = 975.53 (timestamp: 1661701652 -> 2022-08-28T15:47:32)
TEMPERATURE = 30.42 (timestamp: 1661701652 -> 2022-08-28T15:47:32)
PRESSURE = 975.72 (timestamp: 1661701682 -> 2022-08-28T15:48:02)
TEMPERATURE = 30.49 (timestamp: 1661701682 -> 2022-08-28T15:48:02)
PRESSURE = 975.7 (timestamp: 1661701712 -> 2022-08-28T15:48:32)
TEMPERATURE = 30.58 (timestamp: 1661701712 -> 2022-08-28T15:48:32)
```

Figura 2.6: Esempio delle misure raccolte in seguito all'elaborazione dei dati ricevuti da TTN.

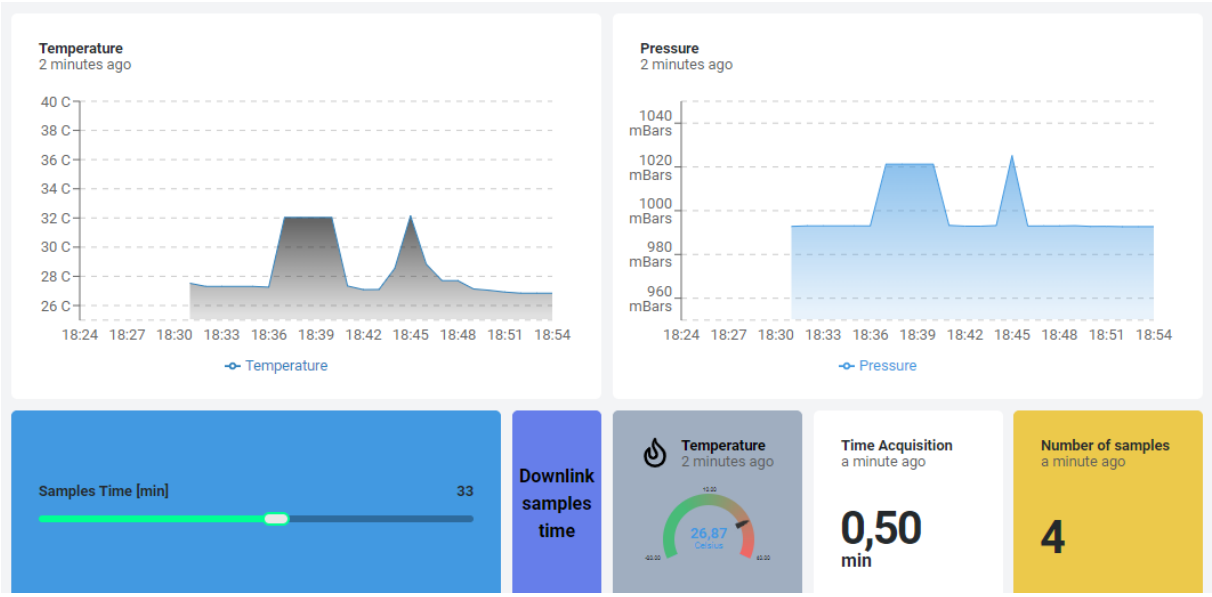


Figura 2.7: Dashboard Datacake.

Capitolo 3

Microcontrollore

3.1 Unità di controllo

In questa sezione viene discussa la scelta e le caratteristiche del controllore che avrà lo scopo di gestire l'acquisizione e la trasmissione dati. Le caratteristiche necessarie che l'integrato deve rispettare riguardano il protocollo di comunicazione *SPI* e un ridotto consumo energetico. Visto che il compito da svolgere non richiede una costante elaborazione dei dati modalità energetiche differenti permetterebbero una maggiore gestione dei consumi.

Per la conoscenza preliminare dell'ambiente *Arduino* e la **non** necessità di particolari prestazioni e/o periferiche si è scelto di utilizzare un microcontrollore della casa produttrice *Atmel Corporation*[1]. L'ambiente di sviluppo *Arduino* permette di programmare molteplici processori con un unico linguaggio "C-*like*" e possiede una notevole *community* di supporto che condivide librerie, consigli e aiuti per tutto ciò che riguarda questo mondo.

A differenza di altri AVR, come per esempio i **megaAVR**, la famiglia *Attiny* ha ridotte dimensioni, ridotto numero di PIN e minore memoria per programmi (memoria *FLASH*) e per variabili (memoria *SRAM*).

3.1.1 Famiglia Attiny

Tra tutte le famiglie di processori disponibili in ambiente *Arduino* la famiglia dei controllori *Attiny* è conosciuta per il suo ridotto consumo energetico. I consumi in questo progetto sono una parte fondamentale del lavoro per questo motivo la suddetta famiglia è stata la prima scelta.

Avendo a disposizione un *Attiny84*, fornito gentilmente dal relatore si è deciso di effettuare i primi test con questa unità. Come tutti i controllori della famiglia anche questo è un AVR a 8-bit basato su architettura RISC. Non esistono PIN dedicati a protocolli specifici, ma alcuni implementano il modulo *USI (Universal Serial Interface)*. Questo modulo

[1] *Atmel* nel 2016 è stata acquisita da *Microchip Tecnology*, ma questi processori sono ancora assegnati sotto il nome della prima.

mette a disposizione le risorse hardware minime per una comunicazione seriale sincrona che permette alte velocità di trasmissione con un minore carico alla *CPU* rispetto a soluzioni basate solo sul software. Tramite *USI* si riesce ad implementare protocolli come I^2C e *SPI*.

A livello di periferiche l'*Attiny84* risulta adatto al controllo del sensore di pressione (ref:1.2) e del modulo LoRaWAN (ref:2.2.2). Anche il numero di pin digitali, nonostante ne siano disponibili solo 8, è sufficiente per comandare i due dispositivi. L'*Attiny84* inoltre possiede al suo interno un oscillatore da 8 MHz che ne permette l'utilizzo privo di componenti esterni.

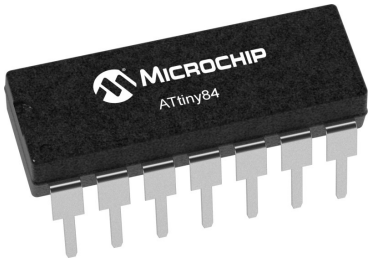


Figura 3.1: Attiny84

Per sfruttare a pieno le potenzialità della rete LoRaWAN nel progetto viene utilizzata la libreria **LMIC** (descritta in seguito nel paragrafo 4.1); l'estensione di questa libreria è visibile anche nella sua dimensione in *FLASH* del microcontrollore che, nonostante le accortezze sull'esclusione di parti non utili all'algoritmo principale, si è rivelata molto onerosa. La memoria per i programmi da 8 kB dell'*Attiny84* è quindi risultata insufficiente. Come potenziale sostituto si è valutato un terminale della stessa famiglia, con almeno 32 kB di FLASH, come per esempio l'*Attiny3224*.

Sfortunatamente l'*Attiny3224* viene realizzato con un *package* che ne consente l'utilizzo solamente saldato su una scheda **PCB**; questo aspetto è facilmente sormontabile, l'inconveniente maggiore si trova nella mancanza di tali dispositivi nel breve termine che quindi ne impedisce l'utilizzo.

La ricerca ha avuto fine con la scelta del più comune *Atmega328p*.

3.2 Atmega328p

La serie *Atmega* è formata da controllori che prevedono un'estensione delle caratteristiche della precedentemente descritta *tinyAVR*. Oltre alla stessa architettura RISC a 8-bit questi controllori vantano maggiore memoria disponibile, un esteso numero di periferiche e generalmente un elevato numero di pin; mantengono inoltre la caratteristica degli AVR di avere ridotti consumi energetici.

3.2.1 Sleep Modes

Gli AVR in generale, e *Atmega328p* in particolare, possiedono diverse modalità di risparmio energetico, dette **sleep modes**. Per ridurre i consumi questi integrati disabilitano determinati moduli in base alla modalità *sleep* scelta. Nel caso specifico dell'Atmega328p ci sono **sei** modalità di "riposo" che si differenziano, oltre ai consumi, rispetto le sorgenti che possono riportare l'integrato allo stato attivo. Gli eventi che possono "svegliare"

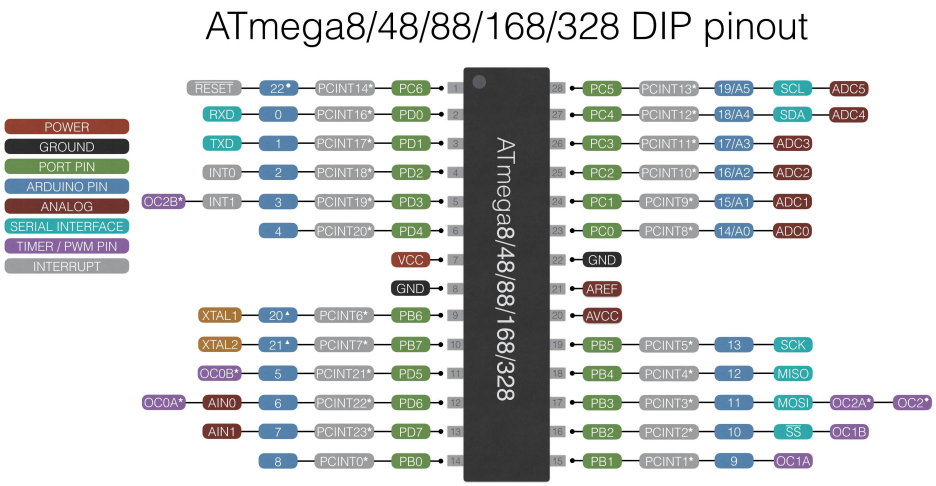


Figura 3.2: Piedinatura dell'integrato Atmega328 in riferimento al package DIP-28

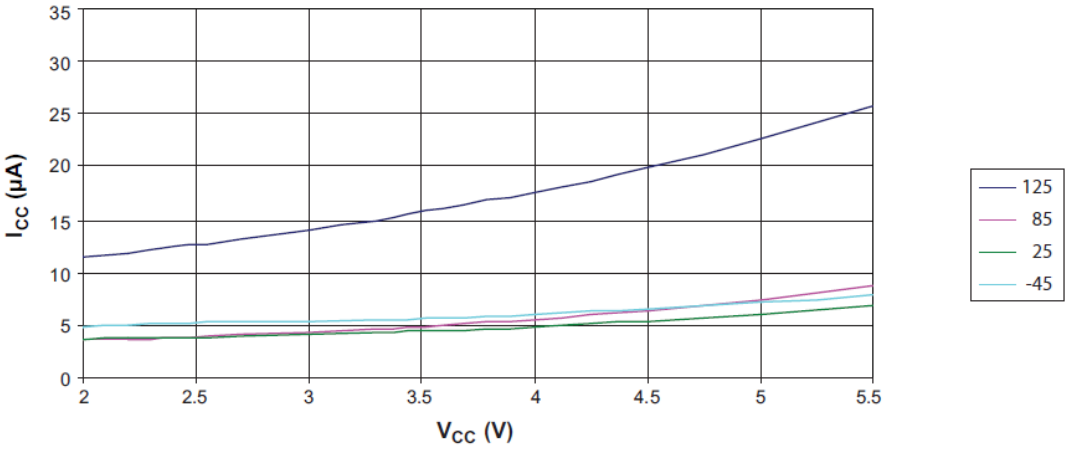
il *micro* possono essere esterni: come *interrupt* esterni, variazioni di pin, interfaccia seriale; oppure eventi interni al micro quali: interrupt interni, *watchdog*, completamento conversioni ADC.

Dalle informazioni riportate nel *datasheet* del dispositivo si è analizzato che lo stato di sleep **Power Down**, con la fase di *wake-up* dettata dal timer **watchdog**; sia la più indicata per la situazione. Ogni volta che il dispositivo non è nelle fasi di trasmissione o misura viene abilitato il watchdog che opera con un clock interno a 128 kHz separato rispetto a quello delle altre periferiche e quindi disabilitabile separatamente rispetto al clock della CPU. Analizzando il grafico (riportato in figura 3.3a) della corrente in ingresso all'integrato in questa modalità si può osservare un consumo tipico di circa 5 μA rispetto ad una temperatura di lavoro di 25 °C e con alimentazione $Vcc = 3.3\ V$. A parità di modalità *sleep* selezionata l'*Attiny84* presenta assorbimenti di corrente paragonabili al "fratello maggiore": figura 3.3b

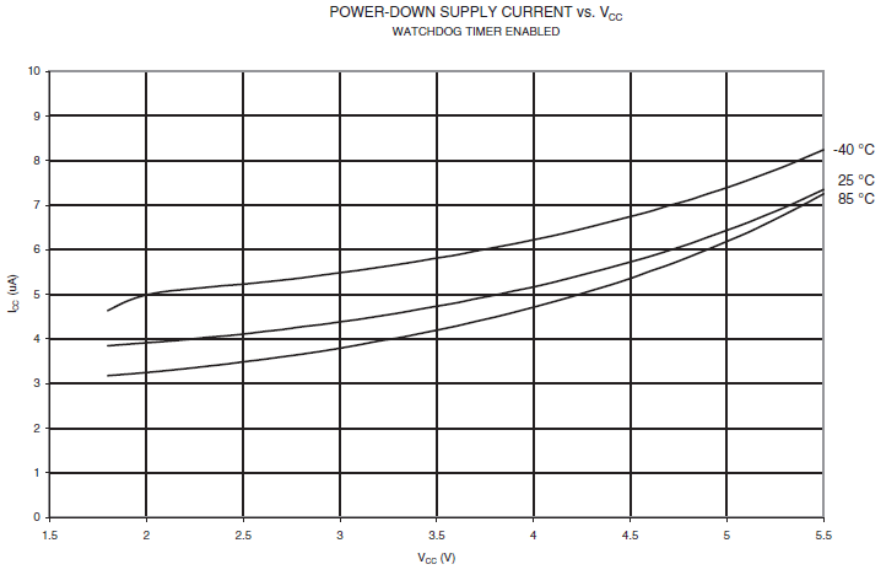
Durante la fase attiva del controllore invece la differenza di consumi tra l'Attiny e l'Atmega diventa maggiore, a discapito di quest'ultimo. Un dispositivo di raccolta dati come quello del progetto passerà la maggior parte del tempo in fase di sleep, riducendo le fasi *active* allo stretto necessario per la misurazione.

Una valida alternativa alla modalità Power Down è la sua versione *Save* in cui l'unica differenza è la possibilità di attivare il timer2 con un oscillatore a bassa potenza esterno. Questa modalità in termini di consumi è molto simile alla precedente, il vantaggio del clock esterno si riscontra nella precisione dei tempi di risveglio: il clock interno del watchdog, come si può osservare in figura 3.3c, è sensibile alla variazione di temperatura quindi la sua frequenza non è costante a 128 kHz causando un'imprecisione nel tempo di sleep. Nel dispositivo in oggetto i tempi di raccolta dati sono importanti ma nell'ordine di grandezza dei secondi; per questo motivo si è deciso di riattivare l'integrato con il watchdog.

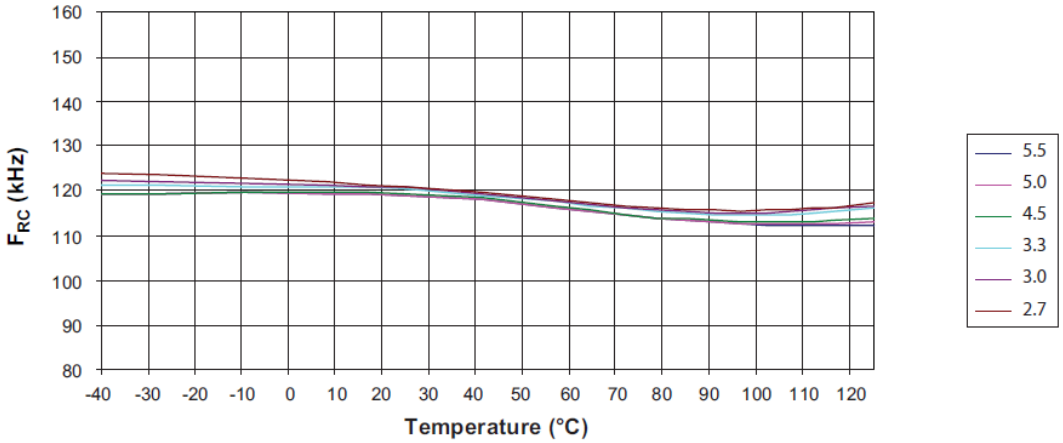
I tempi di ingresso nella *watchdog interrupt subroutine* variano in base al prescaler impostato nel registro *WDTCSR (Watchdog Timer Control Register)* e possono essere impostati da un valore minimo di 16 ms ad uno massimo di 8 s.



(a) Assorbimento di corrente in funzione della tensione di alimentazione Vcc. Riferimento in modalità **Power Down** e watchdog timer abiliato.



(b) Attiny84. Caratteristica di corrente rispetto Vcc, in modalità **Power Down** e watchdog timer abilitato.



(c) Variazione della frequenza di oscillazione del clock interno a 128 kHz.

Figura 3.3: Fonti: datasheet Atmega328p e datasheet Attiny84.

3.2.2 Programmazione Atmega328p

Nella realizzazione del dispositivo viene utilizzato un *Arduino standalone*[2]. Per il controllo completo dell'integrato è necessario modificare il *bootloader* originariamente fornito dalla IDE Arduino, infatti quest'ultimo non permette la personalizzazione dei *Fuses*[3]. Il bootloader altro non è che un pezzo di codice che permette lo scaricamento degli *sketches*. Senza il bootloader per poter inserire del codice nella scheda è necessario un programmatore esterno. Il pacchetto **MiniCore** di *MCUdude* fornisce una serie di librerie che per-

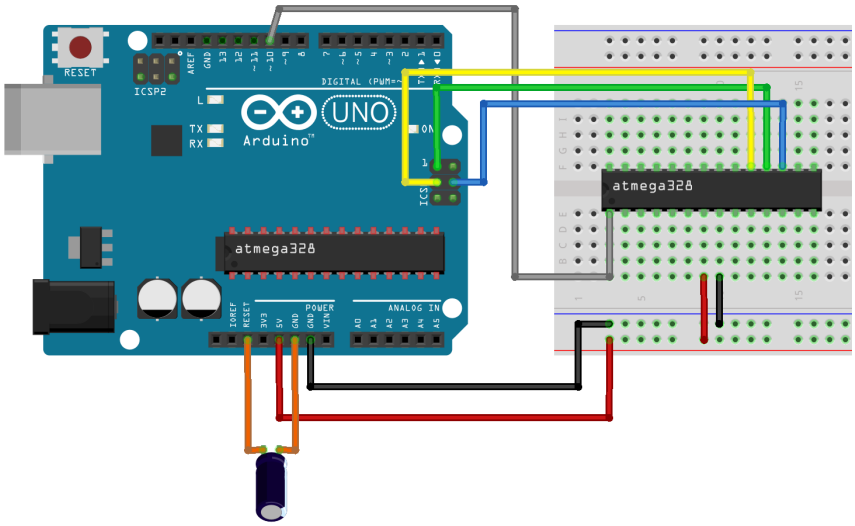


Figura 3.4: Circuito per la programmazione del bootloader.

mettono di estendere il *gestore schede* nativo di Arduino con nuove funzionalità. Questo pacchetto permette, direttamente dall'interfaccia di Arduino, di impostare alcuni parametri del bootloader prima di essere scaricato. La modifica dei *fuses* è un'operazione molto delicata, impostare bits errati potrebbe danneggiare la scheda o impedirne lo scaricamento di nuovi software. Attraverso la gestione della scheda Atmega328p fornita dall'estensione *MiniCore* sono stati impostati due parametri fondamentali:

- **BOD**. Brown-Out-Detection: è un sistema di controllo dell'alimentazione che riavvia il dispositivo nel caso in cui la tensione in ingresso sia insufficiente per il corretto funzionamento del codice e delle periferiche connesse. Nel progetto questa funzionalità viene disabilitata, permettendo una diminuzione dei consumi dato che il relativo modulo viene spento.

- **Clock**. L'integrato viene impostato ad utilizzare come sorgente di clock un oscillatore al quarzo esterno da 16 MHz. L'integrato dispone di un *oscillatore RC* interno a

[2]Per Arduino standalone si intende un integrato utilizzato singolarmente, senza una scheda di supporto per la conversione dei dati seriali in formato *USB*.

[3]I *Fuses* nei microcontrollori AVR sono speciali *bits* che permettono l'attivazione o disattivazione di particolari funzioni. Essi sono modificabili solo attraverso un programmatore e determinano il funzionamento del codice.

8 MHz. Tramite prove sperimentali si è verificato che questa sorgente è troppo imprecisa per le operazioni di trasmissione dati. Per maggiori dettagli si rimanda al paragrafo 5.2.

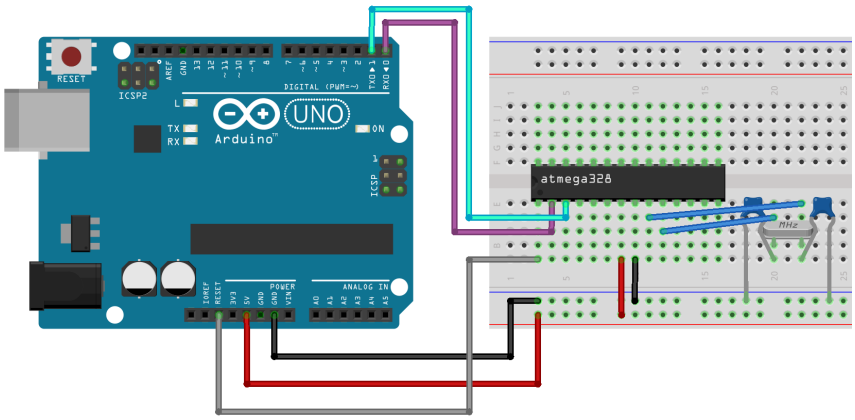


Figura 3.5: Circuito per lo scaricamento dello *sketch*.

Infine per lo scaricamento del software effettivo si è fatto uso di una scheda *Arduino Uno* da cui è stato rimosso l'Atmega328p per sfruttare l'integrato di conversione *Seriale-USB* **Atmega16u2**. Lo scaricamento avviene tramite seriale collegando i *pin 0,1,reset* dell'integrato standalone (figura 3.2) con i *pin 0,1,reset* della scheda Arduino Uno. Il circuito dei collegamenti viene riportato in figura 3.5.

Capitolo 4

Software

4.1 Libreria LMIC

LoRaWAN negli ultimi anni è diventato uno dei protocolli di riferimento per l'IoT; LMIC è frutto di questo sviluppo. LMIC è stata originariamente trasferita su Arduino da *Matthijs Kooijman* e *Thomas Telkamp* ed in seguito **MCCI** ha esteso il supporto ad altre regioni e ne segue l'attuale mantenimento. **LMIC** è la sigla di *LoRaWAN-MAC-In-C* ed è la libreria più completa per Arduino che implementi il livello MAC LoRaWAN (e non solo il livello fisico LoRa). Questa libreria permette di collegare il *device* ai server di rete che forniscono copertura LoRaWAN. Nel caso specifico la libreria viene inizializzata con i codici di attivazione forniti nella *console TTN* (sezione:2.3). Le chiavi così ottenute vengono poi fornite, mediante variabili, al software del dispositivo che le utilizzerà per avviare la comunicazione. Il *Network server* assegna un non-univoco **device address** ad ogni dispositivo in base alla configurazione scelta: l'*end-device* infatti ha due modalità per accedere al server; la scelta va fatta a priori sia perchè le chiavi di attivazione, avendo un ruolo diverso, cambiano sia perchè il software del dispositivo deve eseguire in maniera differente l'accesso al *network*.

- **OTAA Over-The-Air-Activation**. Il dispositivo esegue una procedura di *join* con il server, da cui riceve un `DevAddr` dinamico e le chiavi di sicurezza vengono concordate con il terminale.
- **ABP Activation-By-Personalization**. In questa configurazione l'indirizzo del dispositivo e le chiavi di sicurezza sono pre-configurate durante la registrazione nella piattaforma. ABP presenta lo svantaggio che i dispositivi non possono cambiare provider di rete in autonomia, ma solo previa modifica manuale delle chiavi.

Grazie all'utilizzo di chiavi di rete dinamiche l'attivazione tramite OTAA è più sicura rispetto a quella ABP; i fornitori del servizio TTN infatti consigliano la prima alternativa e così si è fatto.

Scaricando la libreria LMIC, oltre a file di configurazione vengono forniti alcuni esempi di software. Per il codice del progetto si è presa ispirazione dal file "*ttn_otaa.ino*" in cui

vengono riportati i metodi principali per iniziare il collegamento con i server TTN. Nella *repository GitHub* della libreria viene fornita la documentazione necessaria per l'utilizzo. I metodi di LMIC sono stati sviluppati per una gestione ad eventi dell'esecuzione del software, infatti ogni metodo della libreria richiama una funzione *onEvent()* definita dall'utente ed in base all'evento che occorre si possono costruire una serie di azioni. All'accensione, dopo l'inizializzazione del modulo RFM96, i primi eventi saranno relativi allo stabilimento di una comunicazione. Al successo di questa fase il programma può entrare nel suo `loop`. Il metodo *os_runloop_once()* fornito dalla libreria gestisce tutti gli eventi durante la trasmissione dei dati: dall'inizio di trasmissione alla conclusione della fase di ricezione.

Come discusso nel capitolo 2, LoRaWAN opera a frequenze libere da licenza le quali variano a seconda della regione di lavoro. Per il corretto funzionamento della libreria in Europa è necessario impostare come frequenza di base la banda ISM a 868 MHz: questo passaggio può essere fatto inserendo un'istruzione **`define`** nel file "*lmic_project_config.h*" come segue:

```
#define CFG_eu868 1
#define CFG_sx1276_radio 1
```

La seconda istruzione `define` invece specifica quale modulo trasmettitore viene utilizzato. Nel caso di studio il modulo RFM96 implementa il chip *SX1276*.

LMIC fornisce un supporto anche per dispositivi che implementano la **classe B** del protocollo LoRaWAN (2.2.1). I metodi ad esso associati non si intersecano con quelli necessari al funzionamento della **classe A** ma vengono comunque inclusi nel file binario in ingresso al processore, per ovviare a questo problema vengono inserite le seguenti righe di codice nello stesso file di configuazione di cui sopra.

```
#define DISABLE_PING
#define DISABLE_BEACONS
```

Come ultima modifica alla libreria viene assegnata la mappatura dei pin come riportato in tabella 2.1.

```
#define DIO0PIN   15
#define DIO1PIN   16
const lmic_pinmap lmic_pins = {
    .nss = 9,                                   //SPI chip select pin
    .rxtx = LMIC_UNUSED_PIN,                    //unused pin for rx-tx
    .rst = 14,                                  //reset pin of the board RFM
    .dio = {DIO0PIN, DIO1PIN, LMIC_UNUSED_PIN}, //Digital Input Output pins:
                                                //DIO0, DIO1, DIO2 (unused)
};
```

4.2 Struttura dell'algoritmo

Di seguito viene descritta la logica dell'algoritmo per le tempistiche di raccolta dati, di "*sleep*" e di trasmissione.

L'utente può definire a priori i seguenti tempi che corrispondono a due variabili all'interno del software:

- **`trasmissionTime`**: descrive in secondi l'intervallo temporale tra ogni trasmissione.
- **`Tsamples`**: riporta i minuti che devono intercorrere tra una misura e la successiva.

L'invio dei dati è l'operazione più onerosa in termini energetici rispetto alle altre azioni. Generalemente la frequenza di trasmissione sarà un sottomultiplo della frequenza di acquisizione dati.

Le misure vengono trasmesse in blocchi da `Nsamples` campioni; questo numero viene calcolato come:

$$\texttt{Nsamples} = \frac{\texttt{trasmissionTime}}{\texttt{Tsamples}} \tag{4.1}$$

Per la memorizzazione delle misure si fa uso di due *array* (uno per la pressione e uno per la temperatura) con un numero di elementi sufficiente da contenere almeno `Nsamples` valori.

`Tsamples` corrisponde anche al tempo che il controllore rimane nello stato di *sleep*. Come descritto nella sezione 3.2.1 le fasi di sleep vengono temporizzate dal timer watchdog e al massimo l'ingresso nella *subroutine* avviene ogni 8 s. Per `Tsamples` di valore superiore ad 8 s viene eseguito un numero di risvegli pari a $\frac{Tsamples \cdot 60}{8}$. I risvegli vengono conteggiati all'entrata nella *watchdog subroutine* e finché non si è raggiunto il tempo prestabilito il controllore viene riportato in stato di **sleep**. Durante i periodi in cui il controllore è attivo il *watchdog* viene disattivato per ridurne i consumi.

4.2.1 Accorgimenti per la riduzione dei consumi

Per ridurre ulteriormente i costi energetici tutti i moduli di cui non viene fatto uso sono disconnessi. In particolare: il comparatore analogico, l'ADC, il Brown-Out-Detector, i timer interni[1] e appunto il watchdog.

Infine quando si entra in stati di *sleep* tutti i pin dovrebbero essere configurati per ridurre al minimo l'assorbimento di corrente. Per fare ciò è necessario evitare che i pin rimangano ad uno stato flottante oppure che pilotino carichi resistivi. Di conseguenza tutti i pin digitali vengono portati ad un livello logico basso. Per quanto riguarda i carichi connessi al *micro* rimangono i pin della comunicazione *SPI* i quali, prima di essere impostati a valori da minimo assorbimento, vengono disattivati dal funzionamento *SPI*.

[1] Il metodo `millis()` fornito da Arduino utilizza il timer0 dell'*Atmega328p* per entrare ogni millisecondo nella subroutine. Nella modalità sleep scelta per il progetto tutti i moduli legati alla sorgente di clock principale vengono disabilitati ma per differenti *sleep mode* la *subroutine* del millis risveglia l'integrato ogni *millisecondo*.

```
SPI.end();

pinMode(13, OUTPUT);digitalWrite(13, LOW);       //SCK
pinMode(11, OUTPUT);digitalWrite(11, LOW);       //MOSI
pinMode(12, INPUT_PULLUP);                       //MISO

//RFM96 pins
pinMode(DIO0PIN, OUTPUT);digitalWrite(DIO0PIN, LOW);    //DIO0
pinMode(DIO1PIN, OUTPUT);digitalWrite(DIO1PIN, LOW);    //DIO1
pinMode(lmic_pins.nss, OUTPUT);digitalWrite(lmic_pins.nss, HIGH);
pinMode(lmic_pins.rst,INPUT_PULLUP);    //reset pin RFM96

pinMode(MS5803_CS_PIN,INPUT_PULLUP);  //chip select MS5803
```

Capitolo 5

Consumi e alimentazione

5.1 Tensione di alimentazione

La scelta della tensione di alimentazione viene fatta in base ai componenti in uso e al minimo valore per avere un ridotto consumo di potenza. Vengono di seguito riportati i rispettivi range di lavoro che i tre componenti permettono:

- L'*Atmega328p*, come riportato nel datasheet, accetta in ingresso una tensione nel range: $2.7 \div 5.5$ $[V]$.
- Il sensore barometrico *MS5803* invece ha un range di *performance* tra $1.8 \div 3.6$ $[V]$.
- Infine il modulo RFM96 ha un range operativo tra $1.8 \div 3.7$ $[V]$.

L'opzione iniziale per alimentare il circuito è tramite regolatore di tensione della categoria **78Mxx**, per ridurre il numero di componenti viene scelta un'unica tensione che sia adatta a tutti e tre i moduli di cui sopra. La tensione inizialmente scelta è *3,3 V*; il regolatore in questo caso sarebbe il *78M33*, montato come in figura 5.1.

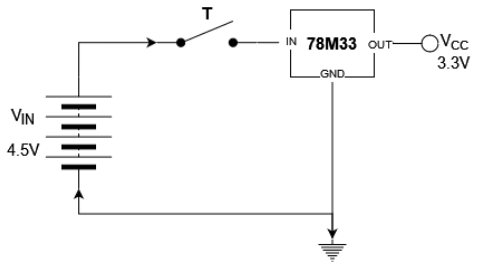


Figura 5.1: Circuito di alimentazione

Durante la fase di test in cui sono stati raccolti i dati di assorbimento di corrente da parte dell'intero apparato si è osservato che il regolatore di tensione assorbe una corrente *quiescente* superiore a 2 mA. In questa configurazione il lavoro di riduzione al minimo dei consumi da parte del controllore viene vanificato.

Una possibile alternativa al regolatore *78M33* è l'utilizzo di convertitori step-up DC/DC come l'integrato *LTC3525-3.3* il quale fornisce in output una tensione di 3.3 V con la particolarità di avere una corrente quiescente nell'ordine dei *microAmpere*. Per ovviare al problema nel progetto si è pensato di semplificare e collegare direttamente la batteria alla linea d'alimentazione del circuito. Per facilitare la sostituzione in caso di esaurimento

è opportuno l'utilizzo di batterie comuni, facili da reperire; per esempio una coppia di batterie stilo *AA* in serie in modo da ottenere 3 V.

La scelta di **non** fare uso di regolatori di tensione comporta una maggiore fluttuazione all'ingresso dei componenti a causa delle condizioni delle batterie. Per semplicità in seguito si assume il pacco batterie come un generatore di tensione ideale: mantiene costante la sua tensione per tutta la durata di vita, non vi è influenza dalla temperatura e inoltre si assume che l'energia totale fornita sia costante rispetto alla corrente erogata.

Oltre al controllore Atmega anche il sensore MS5803 e il modulo RFM96 dispongono di modalità di risparmio energetico e vengono attivate al termine delle rispettive azioni. Questa possibilità unita al fatto che in queste modalità i rispettivi consumi sono estremamente ridotti (nello stesso ordine di grandezza del controllore) **non** si è reso necessario l'utilizzo di una coppia di transistor per l'accensione controllata dei dispositivi.

5.2 Sorgente di clock

La scelta della frequenza di lavoro è fondamentale per ottenere buoni risultati. L'*Atmega328p* dispone di un oscillatore RC interno da 8 MHz che ne permette l'utilizzo privo di componenti esterni riducendo costi e consumi. Tramite misure sperimentali il clock interno è risultato essere troppo sensibile a variazioni di temperatura e tensione. La mancanza di una temporizzazione affidabile provocava instabilità nella connessione al server TTN e la mancata ricezione di messaggi di **downlink**.

Come si può osservare dal grafico tratto dal *datasheet* e riportato in figura 5.2 la diminuzione della frequenza operativa durante la fase *active* può essere circa $\frac{1}{6}$ tra l'utilizzo, come in progetto, di un clock da 16 MHz e quello di uno da 1 MHz. Questa differenza si riduce quando l'integrato va in fase di *sleep* infatti durante questo periodo il CLK_{CPU} è disattivato mentre rimane attivo il watchdog timer.

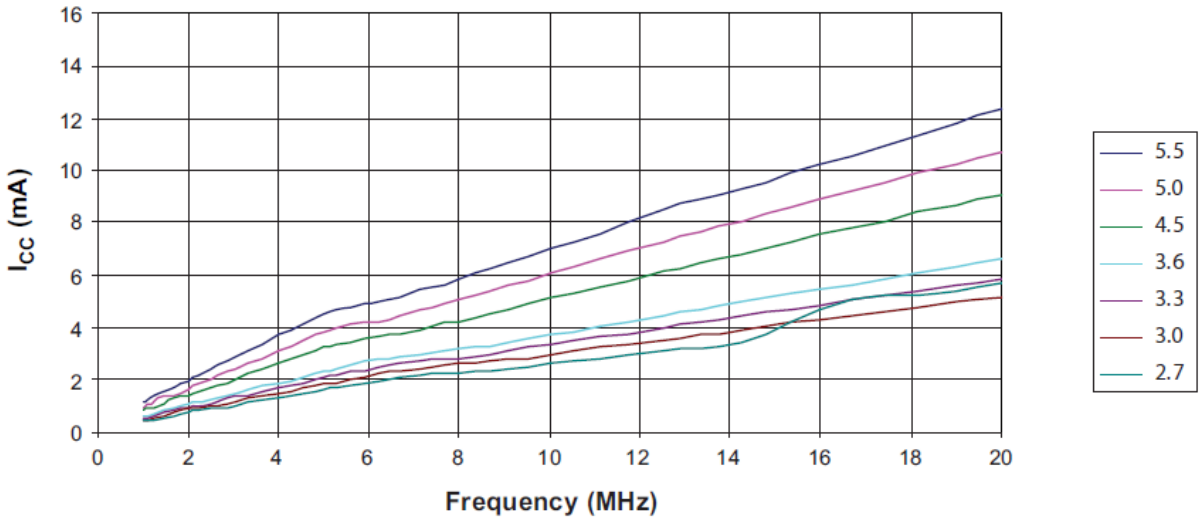


Figura 5.2: Corrente assorbita da *Atmega328p* rispetto alla frequenza del clock. Come riportato a lato del grafico le misure sono state fatte a differenti valori di V_{cc}.

Per migliorare ulteriormente l'autonomia, con l'utilizzo di un quarzo **esterno** da 1 MHz, si trova il compromesso ottimale tra precisione e consumi. La mancata reperibilità ne ha impedito l'utilizzo per il progetto e per test sperimentali.

5.3 Test

Lo studio dei consumi fa riferimento alle condizioni di test; la frequenza di acquisizione dati, l'intensità del segnale e la distanza dai gateway sono parametri che possono variare con l'utilizzo e determinano una fluttuazione dei consumi. In base ai dati raccolti con le prove si può dedurre il consumo medio orario del complesso. Nella valutazione viene esclusa la fase di *join* alla rete perché avviene solamente all'accensione.

L'esecuzione delle misure viene effettuata tramite scheda di acquisizione **DAQ** di *National Instrument* mediante misura di tensione su resistenza di *shunt*. La differenza di tensione ai capi della resistenza viene poi messa in rapporto al valore della resistenza stessa per ottenere l'andamento della corrente in ingresso al circuito.

Le prove sperimentali sono state svolte con trasmissioni ogni *due* minuti e misure ad intervalli di *30 secondi*. L'andamento della corrente viene riportato in figura 5.3 da cui si possono osservare i 3 *picchi* della fase di lettura (la quarta lettura avviene in successione alla trasmissione, quindi non presenta uno *spike* isolato) e l'intervallo di tempo in cui il controllore trasmette i dati e aspetta il termine della finestra di ricezione.

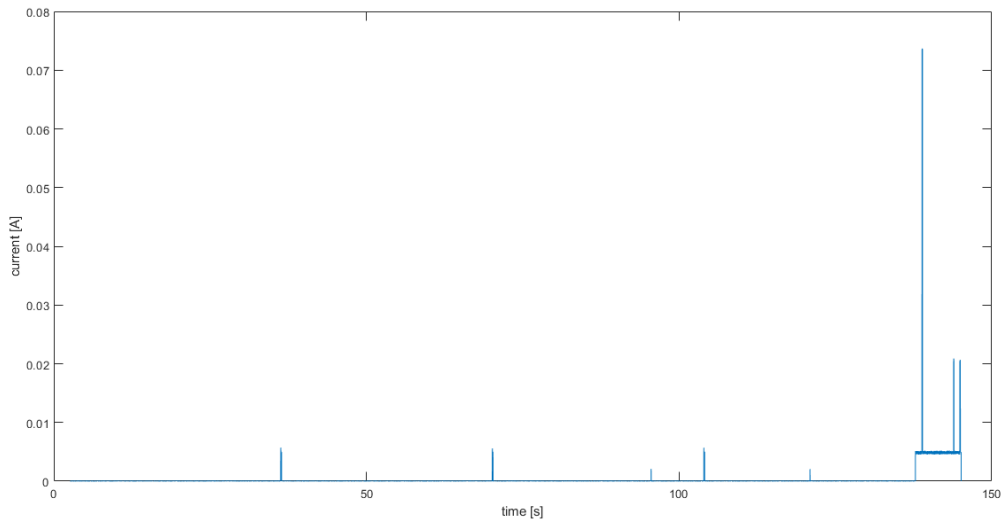


Figura 5.3: Assorbimento di corrente del dispositivo.

Il risultato viene in seguito esteso ad 1 ora di utilizzo in maniera empirica: si mantengono 4 misure e 1 trasmissione ma si estende il tempo inserendo i valori dello stato di riposo dell'intero dispositivo. Il consumo di energia viene calcolato per via grafica come il prodotto dell'area sottesa alla curva della corrente e la tensione di alimentazione cioè 3 V. Infatti:

$$E[J] = \int_{t1}^{t2} p(t)dt = \int_{t1}^{t2} v(t) \cdot i(t)dt = Vcc \cdot \int_{t1}^{t2} i(t)dt$$

Attraverso la funzione **`trapz`** di *Matlab* viene calcolata l'area di interesse e l'**energia** che ne risulta vale: $E = 44\ J$. Quindi in un'ora di utilizzo il dispositivo nel suo complesso consuma $44\ J$.

In base all'energia disponibile nella batteria si può stimare l'autonomia del sistema di misura. Preso come riferimento una coppia di batterie AA da *2500 mAh*, la serie di queste offre una tensione di 3 V a pari capacità. L'energia erogabile dalle batterie risulta: $E_{batt}[Wh] = 2.5Ah \cdot 3V$. Riportando in *Joule*: $E_{batt}[J] = (2.5 \cdot 3) \cdot 3600 = 27000J$.

L'autonomia stimata in giorni vale:

$$autonomia = \frac{E_{batt}}{E \cdot 24} = 25.5 \quad giorni.$$

Conclusioni

L'obiettivo del progetto era fornire a ricercatori che studiano la qualità delle acque un sistema autonomo di raccolta dati. Il progetto viene sviluppato per poter essere esteso con ulteriori sensori ad esempio di conducibilità e torbidità in modo da aumentare la completezza delle informazioni raccolte.

I dati sono accessibili all'utente finale tramite la dashboard Datacake dalla quale è inoltre possibile inviare comandi di reset ed è già predisposto l'algoritmo di invio di messaggi di modifica dei tempi di raccolta e trasmissione dati. I messaggi di downlink sono facilmente soggetti a mancate ricezioni; per cui al momento non vengono implementati nella versione corrente del software. La possibilità di avere un controllo remoto facilita la risoluzione dei problemi e la versatilità delle misure.

Il *device* ha richiesto un'analisi approfondita di controllori *low power*, lo studio di un nuovo protocollo di comunicazione e della struttura di rete. La scelte fatte hanno portato al compimento di quelli che erano gli obiettivi iniziali. Durante il percorso sono state notate migliorie che per determinati problemi non hanno avuto impiego. Il dispositivo può essere utilizzato per migliorare il monitoraggio ambientale; fornendo, si spera, una migliore qualità del bene primario su cui si basa la vita.

Appendice A

TTN Payload formatter

Di seguito viene riportato il codice in *Javascript* del payload inserito nel server TTN.

```
function decodeUplink(input) {
  var Nsamples = 0;
  var TimeToSleep = 0;
  var transmissionTime = 0;
  var i = 0;
  var pres = [];
  var temp = [];
  if(input.bytes[i] === 0x40){      //@
    transmissionTime = (input.bytes[1]<<8 | input.bytes[2]);
    TimeToSleep = input.bytes[3];
    Nsamples = (transmissionTime)/(TimeToSleep*60);
    i = 4;
  }
  if(input.bytes[i] === 0x50){      //'P'
    i+=1;
    while (i < (Nsamples*3)){
      pres.push((input.bytes[i]<<16|input.bytes[i+1]<<8 input.bytes[i+2])/100);
      i += 3;
    }
  }
  if(input.bytes[i] === 0x54){      //'T'
    i += 1;
    while (i < (Nsamples*5 + 2)){
      var ib = input.bytes[i];
      if(ib & 0x80 === 0x80){
        temp.push(-1*((ib<<8 | input.bytes[i+1])-0x8000)/100);
      }
      else{
```

```
        temp.push((ib<<8 | input.bytes[i+1])/100);
      }
      i += 2;
    }
  }
  return {
    data: {
      pressure : pres,
      temperature : temp,
      timeacq : TimeToSleep,
      nsamples : Nsamples
    }
  };
}
```

Appendice B

Datacake Payload formatter

Viene qui riportato il codice Javascript del decoder di Datacake.

```
function Decoder(bytes, port) {
    var ts = measurements.TIMEACQ.value;
    var Nsamples = measurements.NSAMPLES.value;
    var tsTTN = rawPayload.uplink_message.decoded_payload.timeacq * 60;
    var nsTTN = rawPayload.uplink_message.decoded_payload.nsamples;
    if(tsTTN !== 0 && nsTTN !== 0){
        ts = tsTTN;
        ns = nsTTN;
    }
    var now = Date.now() /1000;     //ms of UNIX time converterd to seconds
    for(var k=0;k<Nsamples;k++){
            var before = ((Nsamples - k) * ts);
            decoded.push({
                "field": "PRESSURE",
                "value": rawPayload.uplink_message.decoded_payload.pressure[k],
                "timestamp": (now - before)
            });
            decoded.push({
                "field": "TEMPERATURE",
                "value": rawPayload.uplink_message.decoded_payload.temperature[k],
                "timestamp": (now - before)
            });
    }
    decoded.push({
            "field": "TIMEACQ",
            "value": ts / 60
        });
    decoded.push({
```

```
            "field": "NSAMPLES",
            "value": Nsamples
        });
    return decoded;
}
```

Bibliografia

[1] Pressione dell'acqua
http://www.crestsnc.it/divulgazione/media/idropdf/testo03.pdf

[2] Pressione atmosferica
https://it.wikipedia.org/wiki/Pressione_atmosferica

[3] Dispositivi MEMS
https://it.wikipedia.org/wiki/MEMS

[4] Protocollo SigFox
https://www.sigfox.com/en/what-sigfox/technology
https://build.sigfox.com/sigfox

[5] NB-IoT
Tecnologia NB-IoT e applicazioni

[6] Ricevitore SX1276, chip montato su scheda RFM96
https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf

[7] Caratteristiche LoRaWAN
https://lora-alliance.org/about-lorawan/
http://alessandroblason.it/lorawan-lpwan-868mhz/

[8] Arduino e LoRa
https://docs.arduino.cc/learn/communication/lorawan-101

[9] Libreria LoRa LMIC
https://github.com/mcci-catena/arduino-lmic

[10] Documentazione dashboard Datacake
https://docs.datacake.de/lorawan/lns/thethingsindustries
https://docs.datacake.de/lorawan/downlinks

[11] Atmega MiniCore
https://github.com/MCUdude/MiniCore

[12] Datasheet Attiny84
Microchip device Attiny24-44-84 datasheet

[13] Datasheet Atmega328p
Microchip datasheet Atmega328p

Ringraziamenti

Desidero ringraziare il relatore Alessandro Pozzebon per il supporto, le idee e la pazienza nello sviluppo di questo lavoro.

Ringrazio inoltre i ricercatori Matteo Nigro e Marco Luppichini per la loro disponibilità, per l'aiuto nella ricerca dei sensori e per le informazioni dettagliate del dispositivo da realizzare.