SAE 5IOM10

Qualifier les architectures de réseaux hybrides dédiées à l'IoT

2024/2025

IUT BLOIS

Table des matières

Introduction 3
1.Présentation des Capteurs et ESP 32 4
1.1. Technologie 4
1.1.1. Capteur KY-012 4
1.1.2. Capteur KY-013 4
1.1.3. ESP32 5
1.2. Fonctionnement 5
1.2.1. KY-012 (Buzzer) 5
1.2.2. KY-013 (Température) 5
1.3. Protocole 6
1.3.1. KY-012 (Buzzer) 6
1.3.2. KY-013 (Température) 6
1.4. Montage 7
2. Technologie de communication 7
2.1. Affichage ESP 32 et Thingspeak 7
2.2. RSSI en fonction de la distance 9
2.2.1. Protocole de mesure 9
2.2.2. RSSI des 3 communications 11
2.3. Consommation d'énergie 15
2.3.1. Protocole de mesure 15
2.3.2. Consommation d'énergie des 3 communications et capteurs 16
3. Système globale 18
3.1. Autonomie 18
3.1.1. Protocole de mesure 18
3.1.2. Autonomie de l'ESP 18
3.2. Sécurité 20
3.2.1. Sécurité pour le BLE (Bluetooth Low Energy) 20
3.2.2. Sécurité pour le Bluetooth classique 20
3.2.3. Sécurité pour le LoRa 21
4. Bilan de comparaison 21
4.1. Portée et performance du signal (RSSI) 21
4.2. Consommation d'Energie 22
4.3. Sécurité 22
Conclusion 22

Table des figures....23
Sources....23
Annexe 1 : Capteurs KY-012 et KY-013....24
Annexe 2 : Code BLE et WiFi....26
Annexe 3 : Code Bluetooth et WiFi....37
Annexe 4 : Code LoRa et WiFi....44
Annexe 5 : Consommation énergétique BLE....52
Annexe 6 : Consommation énergétique Bluetooth....55
Annexe 7 : Consommation LoRa....58
Annexe 8 : Consommation énergétique Wifi....71
Annexe 9 : Consommation énergétique des capteurs....73
Annexe 10 : Fiches solutions....75

Introduction

Dans le cadre de ce projet, l'objectif principal est de mettre en place une architecture IoT utilisant deux technologies distinctes de communication sans fil, à savoir Bluetooth/Wi-Fi, BLE/Wi-Fi et LoRa/Wi-Fi, afin d'explorer et de comparer leurs performances en termes de transmission de données et de consommation énergétique. Pour cela, deux capteurs sont utilisés : le KY-012, un buzzer actif, et le KY-013, un capteur de température basé sur une thermistance NTC.

Figure 1: Système à mettre en place et à étudier

L'ESP 32, chargé de la lecture de la température à partir du capteur KY-013 (température), activera le KY-012 (buzzer) si la température dépasse un seuil prédéfini. Ensuite, cet ESP32 enverra l'état du buzzer et la température au deuxième ESP32. Ce dernier affichera ces informations sur son écran OLED et transmettra les données via le Wi-Fi vers un site web ThingSpeak, permettant ainsi de visualiser et d'analyser les données en temps réel.

Pour mener à bien ce projet, nous avons élaboré un diagramme de Gantt afin d'assurer un suivi global de toutes les étapes.

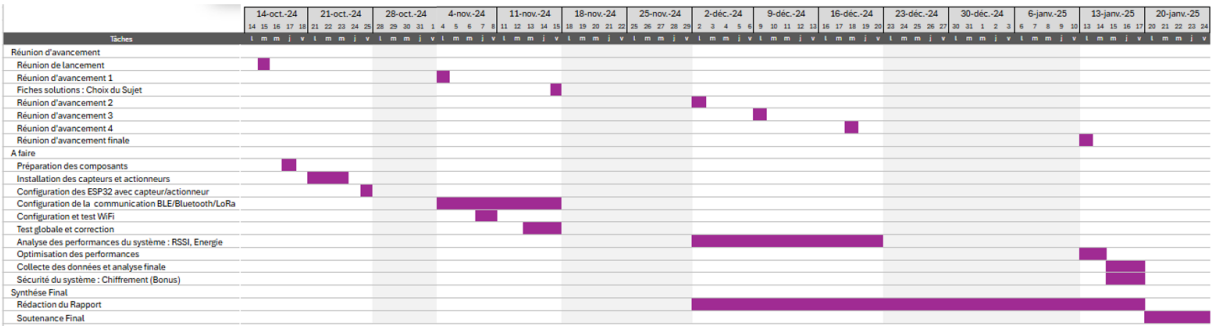


Figure 2: Gantt initiale

Cependant, en raison de contraintes de temps ainsi que de raisons médicales et personnelles, nous avons dû élaborer un second diagramme de Gantt afin de tenter de respecter les délais pour la réalisation du projet.

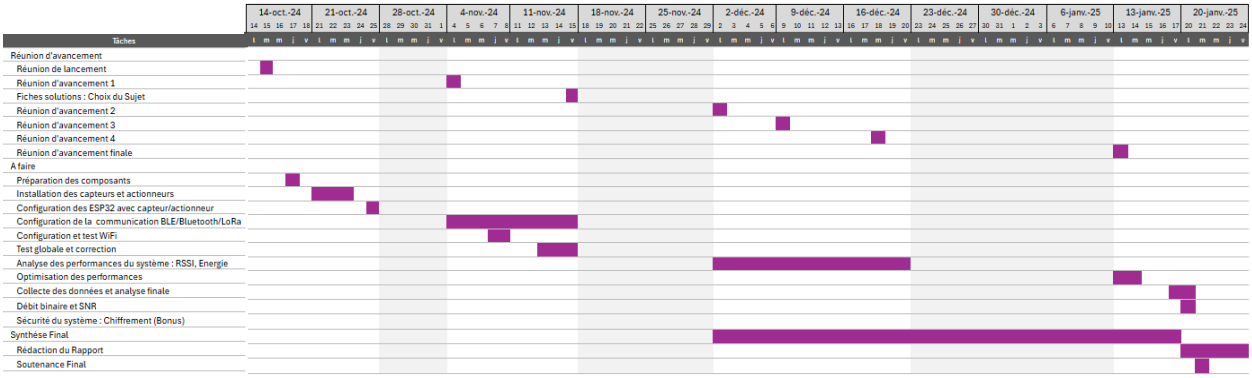


Figure 3: Gantt optimum

1.Présentation des Capteurs et ESP 32

1.1. Technologie

1.1.1. Capteur KY-012

Figure 4: Capteur ky-012

Le KY-012 est un buzzer actif qui utilise la technologie piézoélectrique pour produire des sons. Contrairement aux buzzers passifs, ce module n'a pas besoin d'une fréquence externe pour générer un son. Il est conçu pour produire un signal sonore immédiat dès qu'une tension est appliquée. Ce buzzer génère une fréquence fixe d'environ 2,5 KHz sans nécessiter d'oscillateur externe, grâce à un circuit intégré

1.1.2. Capteur KY-013

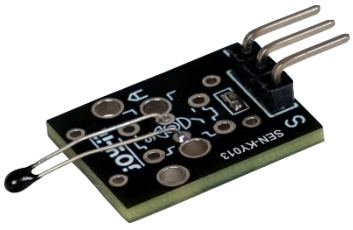


Figure 5: ky-013

Le KY-013 utilise une thermistance NTC (Negative Temperature Coefficient), un type de capteur dont la résistance diminue à mesure que la température augmente. Ce module permet de mesurer des températures comprises entre – 55 °C et + 125 °C. L'avantage principal de ce type de capteur est sa réponse rapide aux variations de température, ce qui en fait un choix idéal pour des applications nécessitant une mesure précise et rapide.

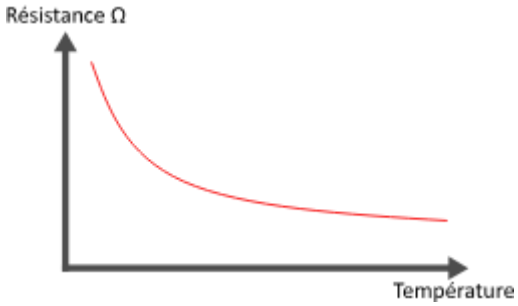


Figure 6: Graphe montrant l'évolution de la résistance en fonction de la température

1.1.3. ESP32

Figure 7: ESP 32 heltec V2

L'ESP32 est une puce microcontrôleur développée par Espressif Systems, idéale pour les projets IoT et embarqués. Elle dispose d'un processeur dual-core à 240 MHz, prend en charge le Wi-Fi et le Bluetooth, et offre de nombreuses interfaces pour la communication avec d'autres composants. Elle est également économe en énergie, sécurisée avec des fonctionnalités comme le chiffrement et le démarrage sécurisé, et facile à programmer via des environnements comme Arduino IDE.

1.2. Fonctionnement

1.2.1. KY-012 (Buzzer)

- **Alimentation et Activation** : Dès qu'une tension d'au moins 3,3 V est appliquée à la broche de signal, le buzzer émet un son. Il fonctionne dans une plage de tension de 3,3 V à 5 V, et la consommation maximale est de 30mA.
- **Fréquence et Sonorité** : Le buzzer émet un son à une fréquence fixe de 2,5 KHz et génère un volume sonore d'au moins 85 dB, ce qui le rend très audible. Il est donc adapté aux applications où un signal sonore puissant est nécessaire.
- **Applications** : Ce type de buzzer est couramment utilisé dans des systèmes d'alarme, des avertissement sonores, ou comme retour sonore dans des appareils électroniques.

1.2.2. KY-013 (Température)

- **Mesure de la température** : La température est mesurée en utilisant un diviseur de tension formé par la thermistance et une résistance fixe connue. Lorsque de la tension

est appliquée au diviseur de tension, la résistance de la thermistance varie en fonction de la température, modifiant ainsi la tension mesurée aux bornes de la thermistance.

- **Calcul de la température** : La relation entre la température et la résistance de la thermistance n'est pas linéaire, mais peut être approximée mathématiquement. À partir de cette variation de résistance, la température peut être calculée. Des formules spécifiques permettent de convertir cette variation de résistance en valeur de température, comme indiqué dans les exemples de code associés au module.
- **Plage de mesure** : Le capteur peut mesurer une plage de températures étendue, de -55°C à +125°C, ce qui permet de l'utiliser dans des environnements très divers.

1.3. Protocole

1.3.1. KY-012 (Buzzer)

- Le KY-012 fonctionne avec un signal numérique : un signal HIGH (actif) génère le son, tandis qu'un signal LOW (inactif) éteint le buzzer.
- Aucun protocole de communication complexe n'est nécessaire. Il ne requiert que la tension adéquate (entre 3,3 V et 5 V) pour s'activer, et est contrôlé directement par un signal numérique envoyé par la broche de contrôle.
- Le capteur est donc simple à intégrer dans les projets électroniques, utilisant un signal numérique pour activer ou désactiver le buzzer.

1.3.2. KY-013 (Température)

- Le capteur KY-013 fournit une lecture analogique de la température, avec une sortie analogique dont la tension varie en fonction de la résistance de la thermistance. Cette sortie peut être lue via une entrée analogique d'un microcontrôleur (comme un ESP32 ou Arduino).
- Aucune communication complexe n'est nécessaire. Il suffit de lire la valeur analogique, puis d'utiliser un calcul mathématique pour obtenir la température en degrés Celsius.
- Le capteur fonctionne donc avec des signaux analogiques et peut être facilement intégré dans des systèmes utilisant des entrées analogiques pour la mesure de température.

1.4. Montage

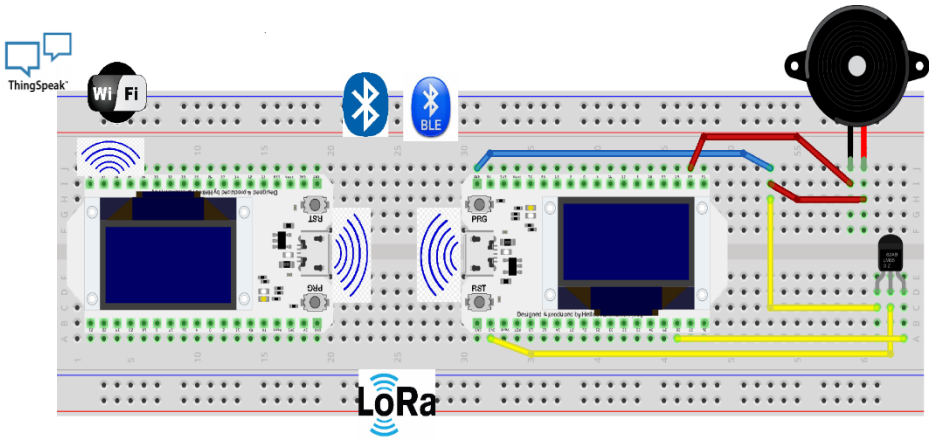


Figure 8: Montage théorique du système avec les capteurs et ESP 32

Voici le montage IoT réalisé avec les 2 capteurs ainsi que les 2 ESP 32 pour ce projet. Il utilise un microcontrôleur connecté à un capteur de température et un buzzer :

- Buzzer : Relié à la broche du pin 13 pour recevoir le signal de commande et à la broche Ground pour la masse. Il émet une alarme sonore en cas de dépassement d'une température critique.
- Capteur de température : Connecté à la broche 4 pour le signal de mesure, à la broche 3,3V pour l'alimentation, et à la broche Ground pour la masse.
- Écran OLED : Affiche en temps réel les données mesurées par le capteur.
- Fonction IoT : Les mesures sont également envoyées à la plateforme ThingSpeak pour un suivi à distance.

Ce circuit permet de surveiller efficacement la température en combinant affichage local et alerte sonore.

2. Technologie de communication

2.1. Affichage ESP 32 et Thingspeak

Nous avons configuré un système pour collecter et afficher des données en temps réel sur la plateforme ThingSpeak. Ce système repose sur l'utilisation d'un capteur KY-013, qui mesure la température ambiante, et d'un microcontrôleur connecté à Internet. Les données collectées sont envoyées toutes les 20 secondes sur ThingSpeak, où elles sont visualisées sous différentes formes.

Figure 9: Affichage sur Oled pour bluetooth

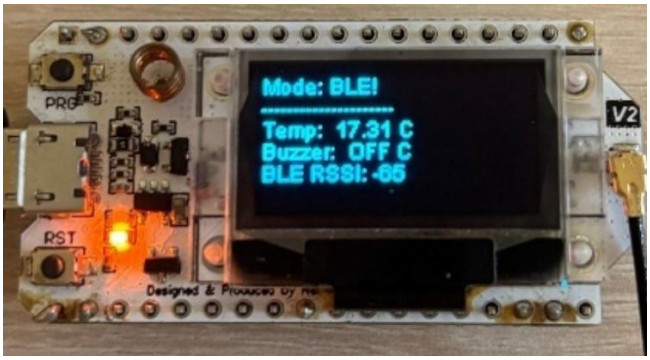


Figure 10: Affichage sur Oled pour BLE

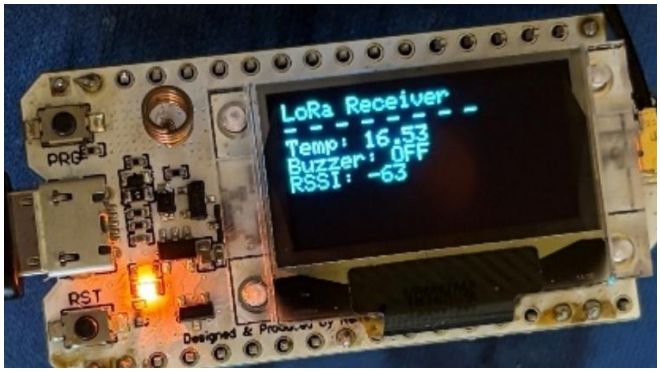


Figure 11: Affichage sur Oled pour LoRa

Voici l'affichage des informations transmises par l'ESP émetteur à l'ESP récepteur. On y trouve les éléments suivants : le mode de communication utilisé pour le projet, la température mesurée par le capteur KY-013, l'état du capteur KY-012, ainsi que le RSSI (indicateur de la qualité du signal) de la technologie concernée.

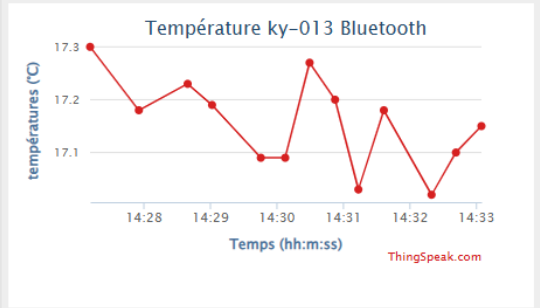


Figure 12: Affichage sur Thingspeak pour Bluetooth

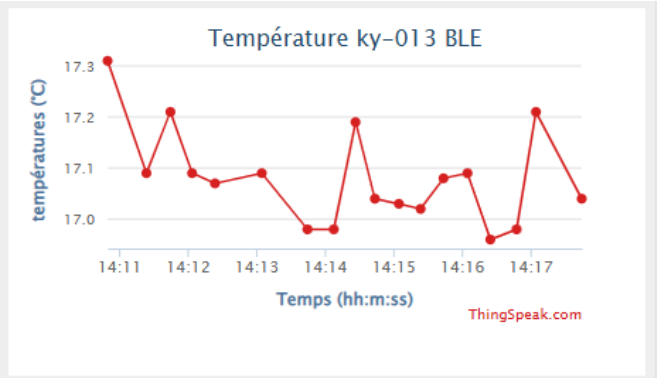


Figure 13: Affichage sur Thingspeak pour BLE

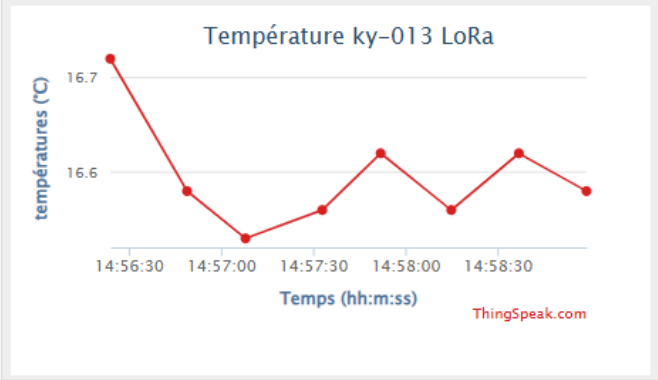


Figure 14: Affichage sur Thingspeak pour LoRa

Les figures ci-dessus présentent un graphique illustrant l'évolution de la température en fonction du temps, permettant de suivre les variations en temps réel ou sur une période spécifique, telles que mesurées par le capteur KY-013.

La configuration a été réalisée à l'aide de scripts qui assurent la collecte des données, leur transfert vers ThingSpeak et leur mise à jour automatique. Ces scripts, sont inclus en annexe pour permettre une reproduction ou une adaptation du système.

2.2. RSSI en fonction de la distance

2.2.1. Protocole de mesure

Pour réaliser les mesures du RSSI, nous avons été contraints trois environnements différents : en intérieur, en milieu urbain et dans un espace libre, tel qu'un champ. Afin de récupérer la valeur de l'atténuation du signal, nous avons intégré une fonctionnalité RSSI dans chaque communication. Voici les plans des zones où les mesures ont été effectuées. Le premier plan représente l'espace libre situé dans les champs à la sortie de Villebarou. Le second montre la zone urbaine, allant de l'IUT à la gendarmerie en longeant l'avenue Maréchale Maunoury. Pour l'intérieur, les mesures ont été réalisés dans le couloir au 4e étage. Dans chaque cas, l'émetteur est resté au point de départ (0 m), tandis que nous avons déplacé le récepteur tous les mètres pour enregistrer les valeurs du RSSI.

Figure 15: Plan pour RSSI dans les champs

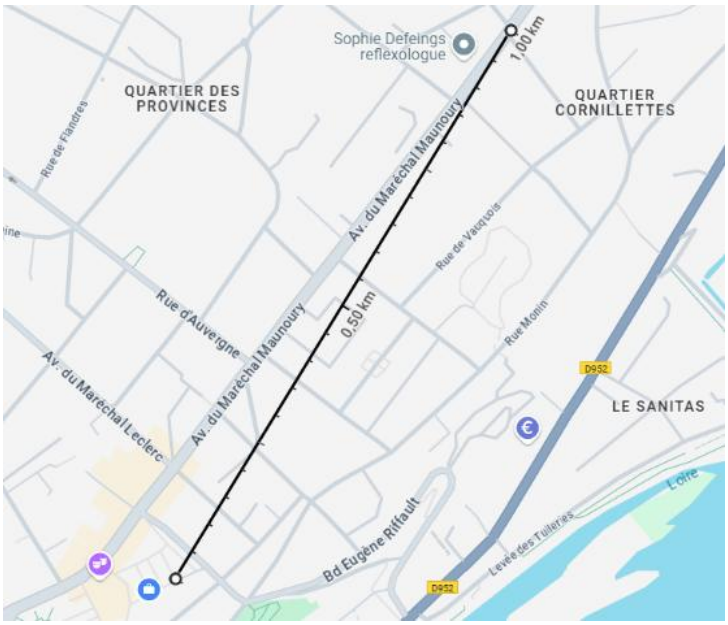


Figure 16: Plan pour RSSI en ville

2.2.2. RSSI des 3 communications

❖ Bluetooth

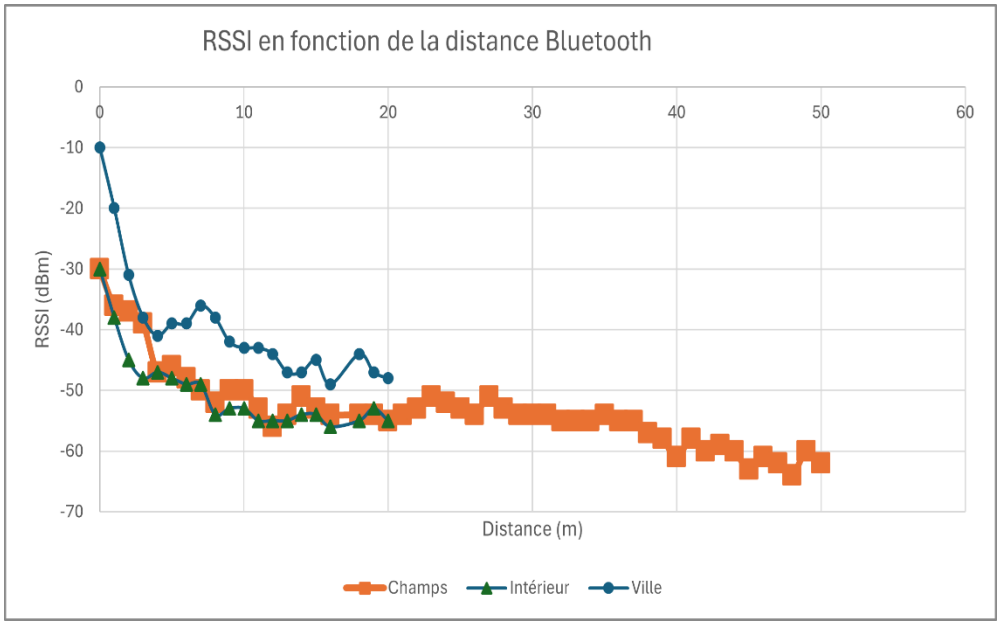


Figure 17: Graphe montrant RSSI en fonction de la distance pour le Bluetooth

L'image montre un graphique représentant la variation du RSSI (Received Signal Strength Indicator) en fonction de la distance pour la communication Bluetooth. Voici les principales caractéristiques du graphique :

- Axe des x (Distance en mètres) : Cet axe représente la distance entre l'émetteur et le récepteur Bluetooth, allant de 0 à 60 mètres.
- Axe des y (RSSI en dBm) : Cet axe indique la puissance du signal reçu (RSSI), exprimée en décibels (dBm).
- Trois séries de données :
    - Champs (orange, carrés) : Correspond aux mesures en environnement ouvert.
    - Intérieur (vert, triangles) : Correspond aux mesures effectuées à l'intérieur d'un bâtiment.
    - Ville (bleu, cercles) : Correspond aux mesures en milieu urbain.

Dans tous les environnements, le RSSI diminue à mesure que la distance augmente, indiquant une perte de puissance du signal. En champs, le RSSI diminue progressivement et reste relativement stable même à des distances plus grandes. En intérieur, le signal est plus faible, à cause des obstacles (murs, ...) qui atténuent le signal. En ville, la courbe montre des variations plus importantes, probablement dues aux interférences et aux réflexions des ondes sur les bâtiments.

❖ BLE

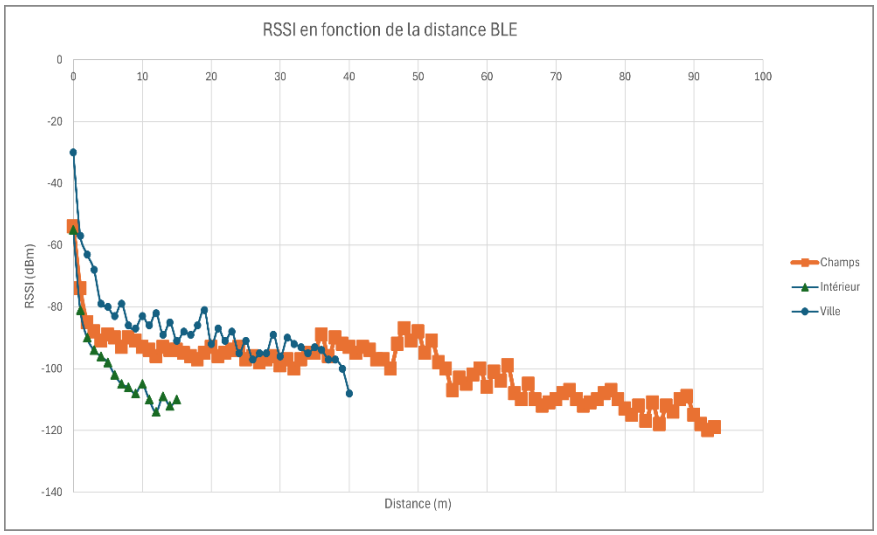


Figure 18: Graphe montrant RSSI en fonction de la distance pour le BLE

L'image montre un graphique représentant la variation du RSSI (Received Signal Strength Indicator) en fonction de la distance pour la communication Bluetooth Low Energy (BLE). Voici une description détaillée :

- Axe des x (Distance en mètres) : Cet axe indique la distance entre l'émetteur et le récepteur BLE, s'étendant jusqu'à 100 mètres.
- Axe des y (RSSI en dBm) : Cet axe mesure la puissance du signal reçu (RSSI), exprimée en décibels milliwatts (dBm).
- Trois séries de données :
    - Champs (orange, carrés) : Mesures en environnement ouvert.
    - Intérieur (vert, triangles) : Mesures effectuées à l'intérieur.
    - Ville (bleu, cercles) : Mesures en environnement urbain.

Le RSSI diminue avec la distance dans tous les environnements. En champs ouverts, le signal reste stable à longue distance (-90 dBm). En intérieur, il chute rapidement (-120 dBm) à cause des obstacles. En ville, il varie de manière irrégulière en raison des interférences. Le BLE est plus affecté dans les environnements encombrés, mais reste relativement stable en champs ouverts, bien qu'atténué à grande distance.

❖ LoRa

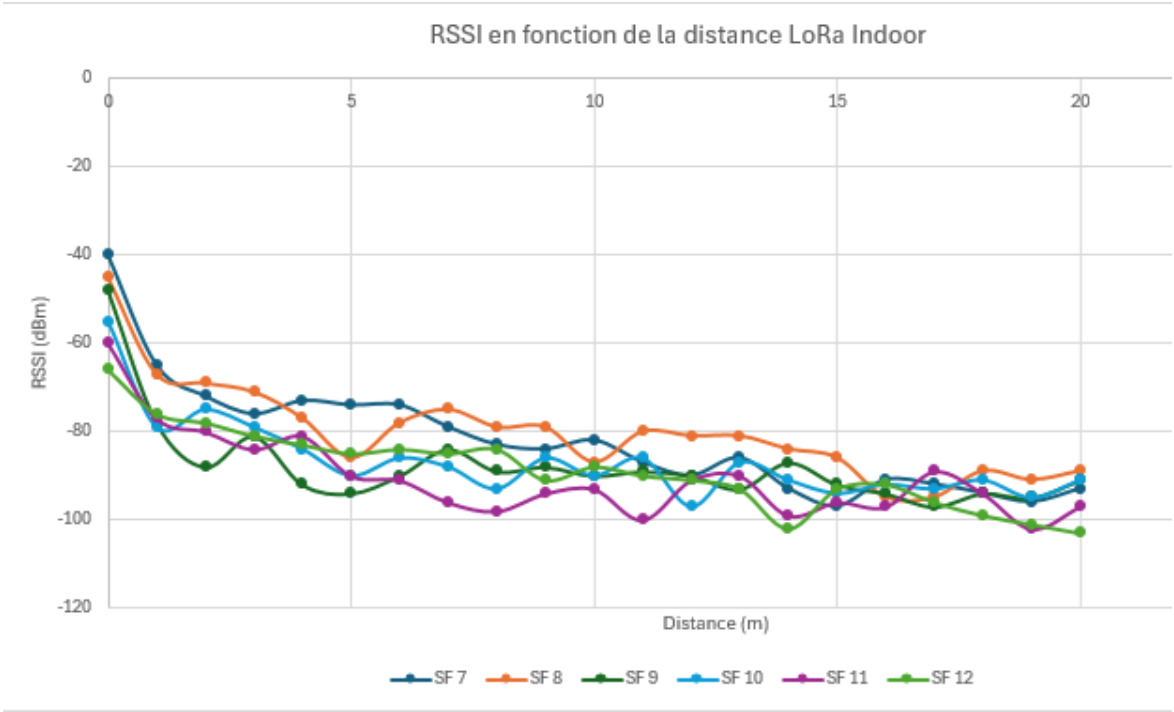
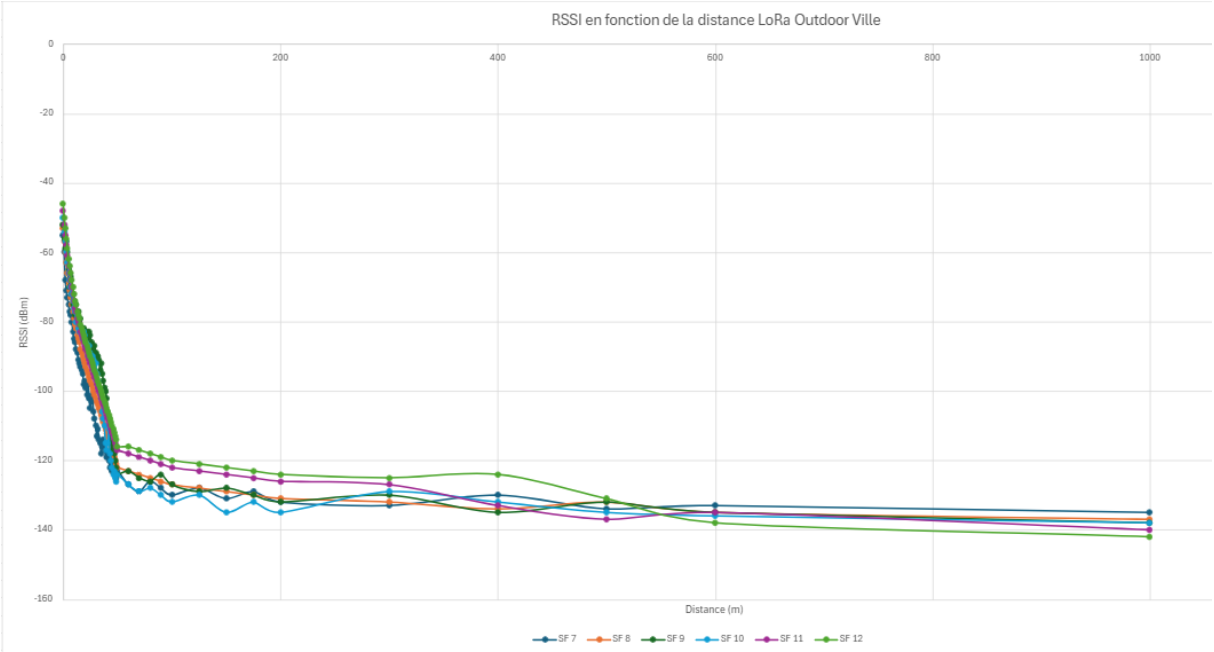


Figure 19: Graphe montrant RSSI en fonction de la distance pour le Lora en intérieur

Le RSSI pour le LoRa en environnement intérieur montre une diminution progressive avec l'augmentation de la distance, mais varie en fonction des Spreading Factors (SF).

- SF 7 et SF 8 : Ces facteurs présentent une meilleure stabilité du RSSI, avec des valeurs autour de -80 dBm sur des distances allant jusqu'à 20 m.
- SF 9 à SF 12 : Les RSSI sont généralement plus faibles, atteignant environ -100 dBm à 20 m. La variation est plus marquée, ce qui pourrait être lié aux interférences et à la sensibilité accrue des SF plus élevés.

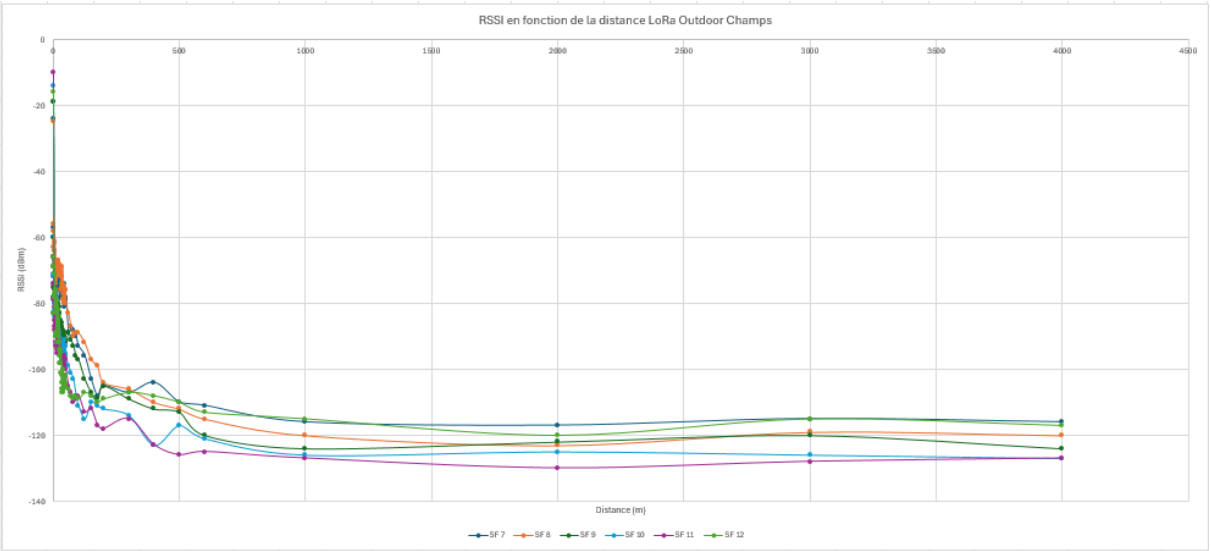
Ainsi, le LoRa en intérieur offre une portée fiable, mais les performances dépendent du choix du SF, avec un compromis entre la portée, la sensibilité et la stabilité du signal.



Le RSSI pour le LoRa en environnement urbain extérieur montre une diminution progressive avec l'augmentation de la distance, mais varie en fonction des Spreading Factors (SF).

- SF 7 et SF 8 : Ces facteurs présentent une meilleure stabilité du RSSI sur de courtes distances, avec des valeurs autour de -40 dBm à -60 dBm jusqu'à environ 200 m. Après cette distance, la diminution devient plus graduelle.
- SF 9 à SF 12 : Ces SF affichent des RSSI plus faibles dès le début, atteignant des valeurs proches de -80 dBm à -100 dBm entre 200 et 1000 m. Les variations sont plus marquées, ce qui pourrait être lié à l'atténuation due aux obstacles urbains et à la sensibilité accrue des SF plus élevés.

Ainsi, le LoRa en ville offre une portée étendue, mais les performances dépendent du choix du SF, avec un compromis entre la portée, la robustesse et la sensibilité au signal dans cet environnement.



Le RSSI pour le LoRa en environnement extérieur rural (champs) montre une diminution progressive avec l'augmentation de la distance, avec des variations en fonction des Spreading Factors (SF).

- SF 7 et SF 8 : Ces facteurs affichent des RSSI relativement élevés sur de courtes distances, avec des valeurs autour de -40 dBm à -60 dBm jusqu'à environ 500 m. La diminution est rapide mais devient plus stable au-delà de 1000 m, atteignant environ -120 dBm.
- SF 9 à SF 12 : Les RSSI pour ces SF sont plus faibles dès les courtes distances, atteignant rapidement environ -80 dBm à -100 dBm entre 500 et 1000 m. Au-delà, ils montrent une meilleure stabilité, convergeant vers des valeurs proches de -120 dBm à -140 dBm.

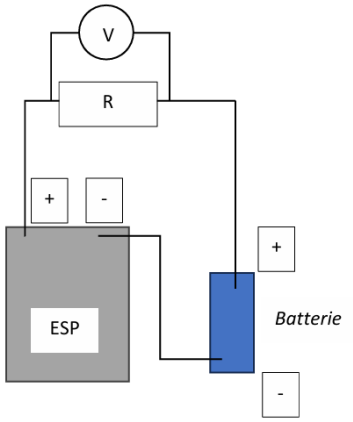
Ainsi, le LoRa en environnement rural offre une portée plus étendue par rapport à l'environnement urbain, avec une atténuation plus progressive grâce à l'absence d'obstacles majeurs. Le choix du SF reste crucial, influençant la portée et la robustesse du signal, avec un compromis entre la sensibilité et la consommation d'énergie.

2.3. Consommation d'énergie

2.3.1. Protocole de mesure

Le protocole de mesure consistait à utiliser une résistance de 0,5 Ω connectée en série avec un ESP et une alimentation de 3,3 V. Un oscilloscope est placé aux bornes de la résistance pour mesurer la tension $U_{mesuré}$. A partir de cette mesure, l'intensité du courant I est calculé en appliquant la loi d'Ohm ($I = \frac{U_{mesuré}}{R}$). Ensuite, la consommation énergétique de l'ESP est déterminée à l'aide de la formule :

$$\text{Consommation énergétique} = \frac{I \times Tension\ d'alimentation}{Temps}$$



2.3.2. Consommation d'énergie des 3 communications et capteurs

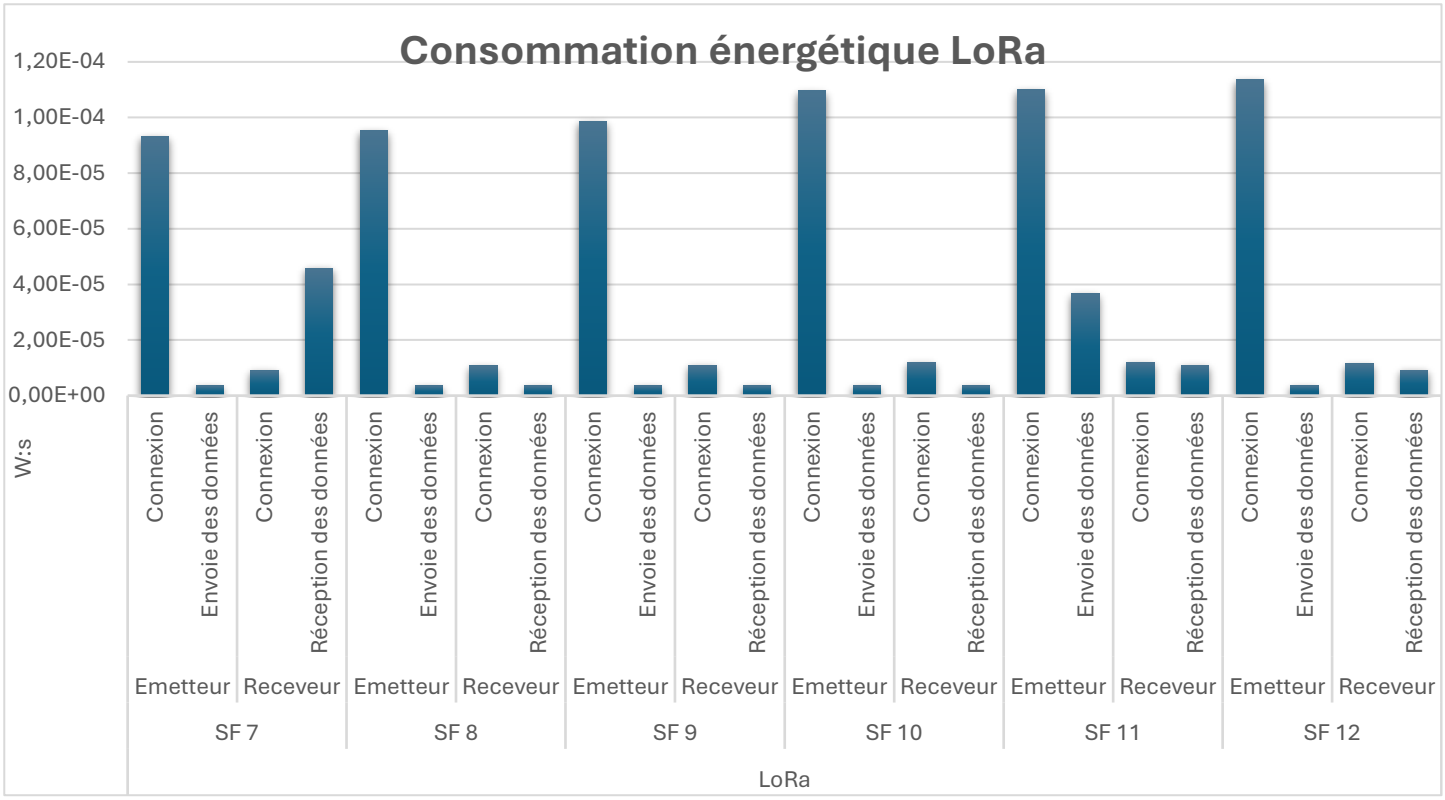


Figure 20: Consommation d'énergie du LoRa sur les SF 7 à SF 12

Le graphique montre que la consommation énergétique des communications LoRa augmente avec les Spreading Factors (SF). Aux SF faibles (SF7 à SF9), la consommation reste modérée, avec des différences minimes entre émetteur et récepteur. Cependant, aux SF élevés (SF10 à SF12), la consommation énergétique, notamment pour l'envoi de données par l'émetteur, devient beaucoup plus importante. Cela souligne le compromis entre portée et efficacité énergétique : les SF faibles consomment moins mais offrent une portée réduite, tandis que les SF élevés, adaptés aux longues distances, nécessitent davantage d'énergie.

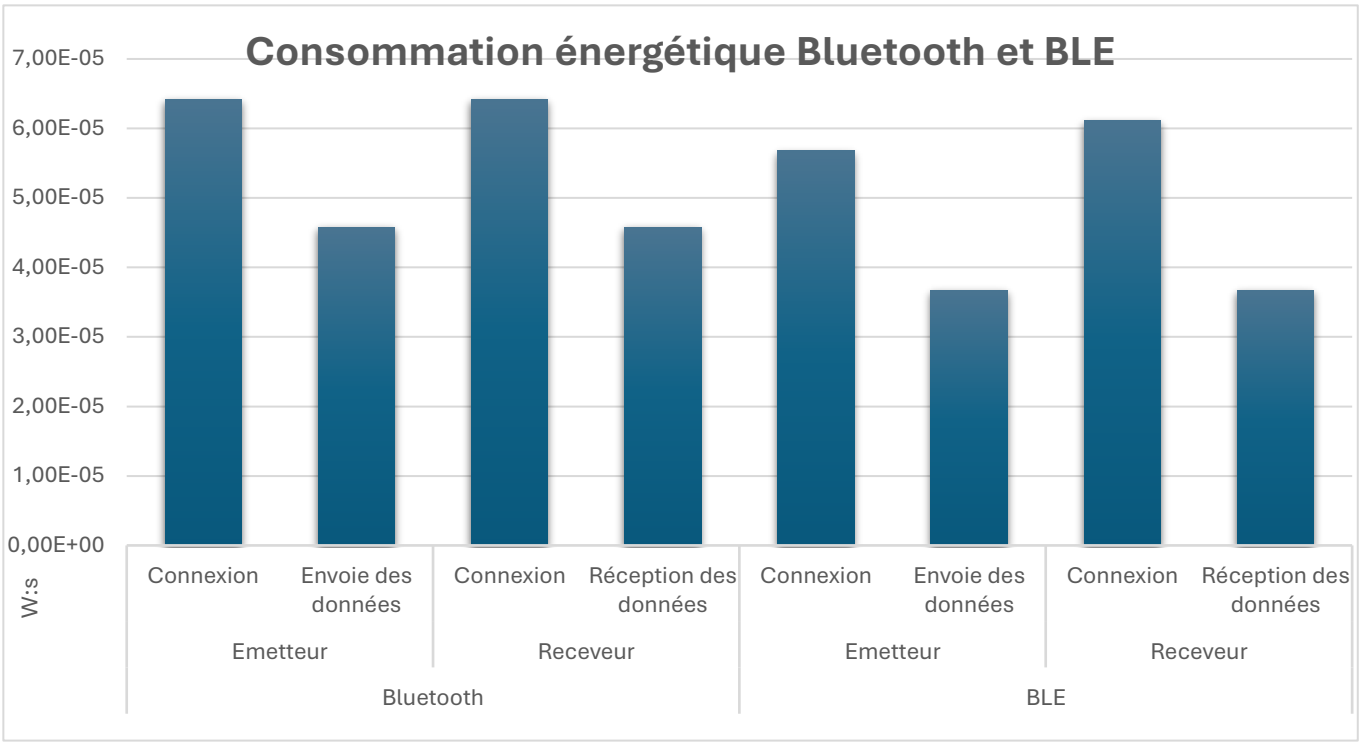


Figure 21: Consommation d'énergie du Bluetooth et du BLE

Le graphique montre que le Bluetooth Low Energy (BLE) consomme moins d'énergie que le Bluetooth classique. La consommation est particulièrement réduite pour l'envoi et la réception des données, ainsi que lors de la connexion. BLE est donc mieux adapté aux applications nécessitant une faible consommation énergétique, comme l'IoT et les dispositifs portables.

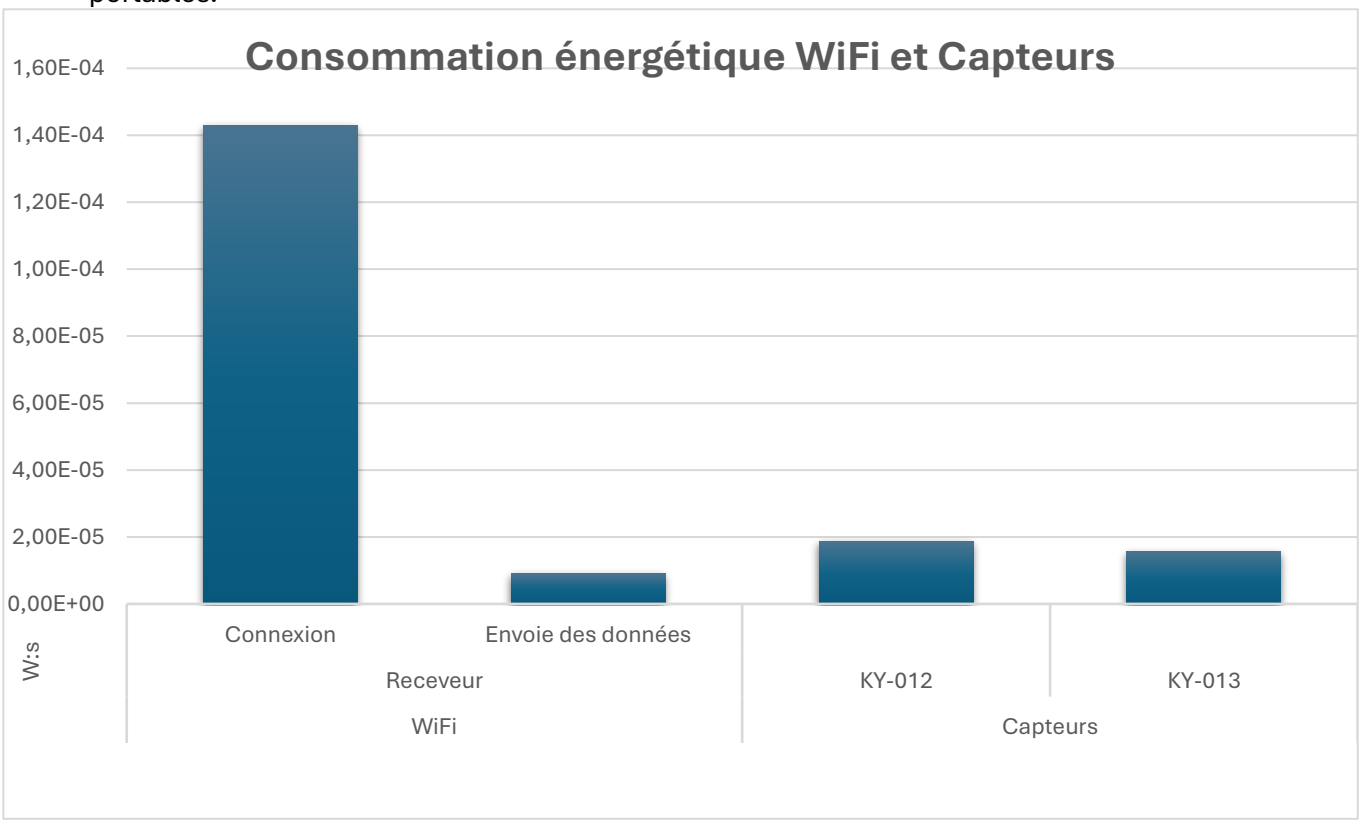


Figure 22: Consommation d'énergie du Wifi et des Capteurs

Le graphique montre que la connexion WiFi consomme le plus d'énergie (~1,4 × 10^{-4} W·s), tandis que l'envoi des données et les capteurs KY-012 et KY-013 consomment beaucoup moins, autour de 6 à 7 × 10^{-5} W·s.

3. Système globale

3.1. Autonomie

3.1.1. Protocole de mesure

La capacité de la pile exprimée en Wh est obtenue en multipliant la tension d'alimentation de la pile (3,3 V) par sa capacité en ampère-heure (Ah). Avec une pile de 1800 mAh (soit 1,8 Ah), la capacité en Wh est :

$$Capacité\ de\ la\ pile\ en\ \left(\frac{W}{h}\right) = 3{,}3V \times 1{,}8Ah = 5{,}94\ Wh$$

La consommation de l'ESP32 peut être mesurée en utilisant un ampèremètre. Celui-ci est connecté en série avec l'ESP32 et la pile pour mesurer le courant consommé en ampères (A). La consommation en watts (W) est ensuite calculée en multipliant la tension d'alimentation (3,3 V) par le courant mesuré.

L'autonomie de l'ESP32 est déterminée en divisant la capacité de la pile (5,94 Wh) par la consommation électrique de l'ESP32 exprimée en watts. Par exemple, si l'ESP32 consomme 0,33 W (3,3 V × 0,1 A), l'autonomie sera de :

$$\text{Autonomie} = \frac{\text{Capacité de la pile } (\frac{\text{W}}{\text{h}})}{Consommation\ de\ l'ESP} = \frac{5{,}94\ Wh}{0{,}33\ W} = 18\ heures$$

3.1.2. Autonomie de l'ESP

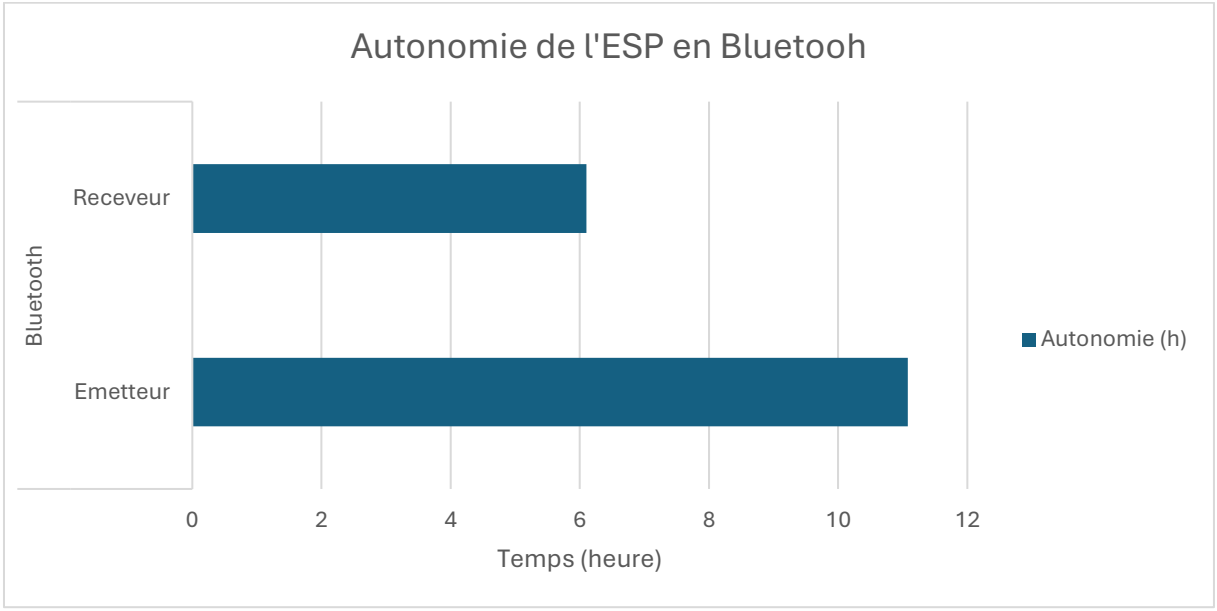


Figure 23: Autonomie de la communication Bluetooth

Le graphique présente l'autonomie de l'ESP en mode Bluetooth pour ses deux fonctions principales : récepteur et émetteur. En mode récepteur, l'autonomie est d'environ 6 heures, tandis qu'en mode émetteur, elle atteint près de 12 heures. Cela montre que la fonction réceptrice consomme moins d'énergie, réduisant ainsi la durée d'utilisation par rapport à l'émetteur, qui est plus économe en énergie.

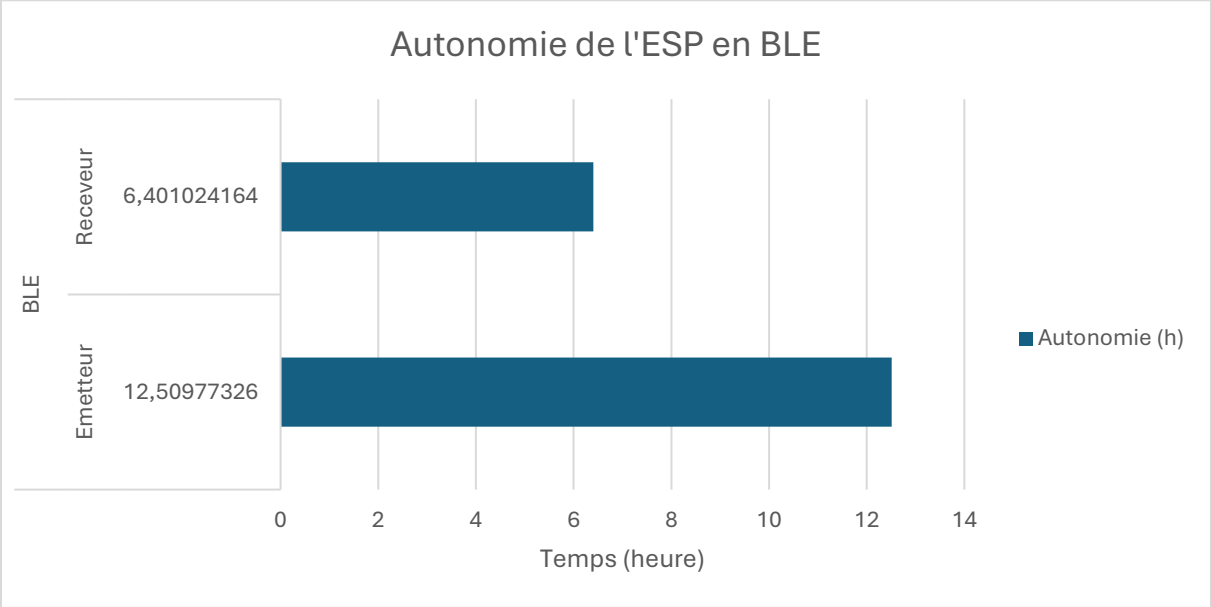


Figure 24: Autonomie de la communication BLE

Le graphique présente l'autonomie de l'ESP en mode BLE (Bluetooth Low Energy) pour les deux fonctions principales : récepteur et émetteur. En mode récepteur, l'autonomie est d'environ 6,4 heures, tandis qu'en mode émetteur, elle atteint environ 12,5 heures. Cela met en évidence que la fonction réceptrice consomme davantage d'énergie, ce qui réduit son autonomie, tandis que le mode émetteur, plus économe, permet une durée d'utilisation plus longue.

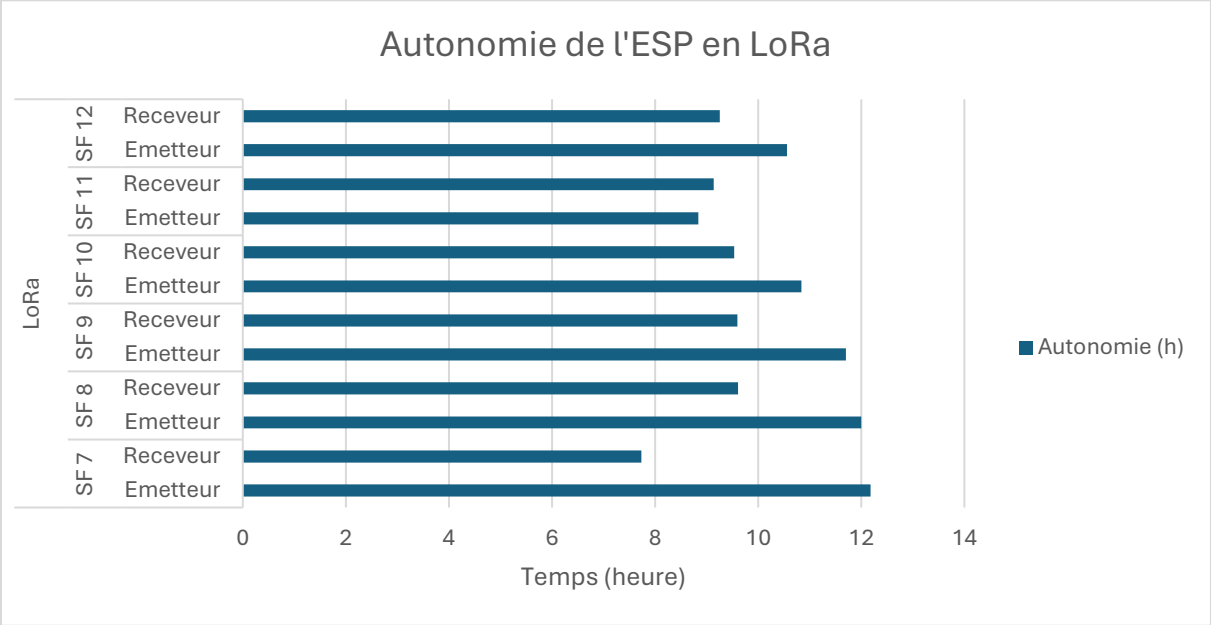


Figure 25: Autonomie de la communication LoRa sur les SF è à SF 12

La figure montre l'autonomie des ESP en communication LoRa pour les Spreading Factors (SF) de SF7 à SF12. L'autonomie décroît au fur et à mesure que le SF augmente, avec une meilleure durée pour SF7. Les émetteurs ont une autonomie légèrement supérieure aux récepteurs.

3.2. Sécurité

Pour sécuriser les communications d'un ESP32 utilisant le BLE, le Bluetooth classique ou LoRa, il est essentiel de mettre en œuvre des mesures adaptées tout en tenant compte de leur impact sur la consommation énergétique.

3.2.1. Sécurité pour le BLE (Bluetooth Low Energy)

Pour renforcer la sécurité des communications BLE, plusieurs actions peuvent être mises en place :

- Utilisation d'un couplage sécurisé : en activant les Secure Connections (ECDH), vous protégez l'échange des clés. Cela augmente légèrement la consommation d'énergie au moment du couplage, mais n'affecte pas les échanges une fois la connexion établie.
- Authentification mutuelle : cette mesure garantit que les appareils validés sont autorisés à échanger des données. L'impact énergétique reste limité puisqu'elle intervient uniquement lors de la connexion initiale.
- Chiffrement des données : activer l'AES-CCM (128 bits) offre une protection efficace contre les interceptions de données. Bien que cela nécessite des calculs cryptographiques, l'impact sur l'autonomie est faible.
- Renouvellement régulier des clés de chiffrement : changer les clés périodiquement réduit les risques liés à une clé compromise. Cette action peut consommer davantage d'énergie à chaque renouvellement, mais reste essentielle pour les sessions longues.

3.2.2. Sécurité pour le Bluetooth classique

Pour le Bluetooth classique, les mesures suivantes permettent de protéger les communications :

- Appairage sécurisé avec un code PIN robuste : un PIN long et complexe renforce la sécurité lors de l'appairage. L'impact énergétique est négligeable, car cette étape est ponctuelle.
- Chiffrement des données transmises : cette mesure protège les données échangées, avec une légère augmentation de la consommation, particulièrement pour les sessions prolongées.
- Mode non découvert : en rendant l'appareil invisible une fois appairé, on réduit les risques d'accès non autorisé tout en diminuant la consommation énergétique, car l'appareil n'émet plus de signaux inutiles.
- Suppression des appareils inutilisés : retirer les appareils appairés non utilisés limite les risques de connexions malveillantes. Cette action n'a pas d'impact sur l'autonomie.

3.2.3. Sécurité pour le LoRa

Pour les communications LoRa, il est crucial de tirer parti des mécanismes intégrés au protocole LoRaWAN :

- Utilisation de clés uniques : configurer des clés spécifiques pour chaque appareil, comme l'AppSKey et la NwkSKey, garantit la confidentialité des données. Cette mesure n'impacte pas significativement l'autonomie.
- Activation par OTAA (Over The Air Activation) : cette méthode génère dynamiquement des clés sécurisées lors de l'appairage. Bien qu'elle consomme légèrement plus d'énergie au moment de l'activation, elle assure une meilleure sécurité.
- Chiffrement des données avec AES-128 : le chiffrement intégré au protocole protège les messages entre les appareils et le serveur. Son impact énergétique est minime.

4. Bilan de comparaison

Le projet de comparaison des technologies de communication sans fil pour l'Internet des Objets (IoT) s'est concentré sur trois protocoles majeurs : Bluetooth Low Energy (BLE), Bluetooth classique, et LoRa. Ces technologies ont été évaluées selon plusieurs critères essentiels, tels que la portée, la consommation d'énergie, la sécurité, et le Received Signal Strength Indicator (RSSI), afin de déterminer leur efficacité dans un environnement IoT typique.

4.1. Portée et performance du signal (RSSI)

Le Bluetooth classique présente une portée modérée, avec un signal qui s'affaiblit rapidement dès 10 à 20 mètres en raison des obstacles physiques et des interférences dans les environnements urbains. En milieu ouvert, la portée est plus stable, mais la performance reste limitée à environ 60 mètres. La diminution du RSSI est bien corrélée à la distance, avec un comportement plus erratique en milieu urbain dû aux réflexions et interférences.

Le Bluetooth Low Energy (BLE), quant à lui, bénéficie d'une portée théorique de 100 mètres dans des conditions idéales. Cependant, en environnement intérieur ou urbain, la portée peut être réduite de manière significative en raison de la faible puissance d'émission et de la susceptibilité aux obstacles et interférences. Le BLE, bien que plus efficace en termes de consommation d'énergie, reste vulnérable aux environnements complexes où le signal peut être fortement atténué.

En revanche, le LoRa se distingue par sa capacité à maintenir une portée impressionnante, même dans des environnements difficiles. En milieu urbain, LoRa montre une stabilité de signal supérieure, avec une diminution progressive du RSSI en fonction des Spreading Factors (SF). Les facteurs SF plus bas (7 et 8) assurent une meilleure stabilité, avec une portée efficace jusqu'à 20 mètres même en intérieur, tandis que des SF plus élevés peuvent réduire la stabilité du signal mais offrir une portée accrue en extérieur.

4.2. Consommation d'Energie

En termes de consommation énergétique, le BLE est l'option la plus économique, conçue spécifiquement pour les applications IoT à faible consommation. Il permet une communication intermittente à faible puissance, ce qui est idéal pour les capteurs alimentés par batterie et les dispositifs nécessitant une longue autonomie.

Le Bluetooth classique, bien qu'efficace pour la transmission de données à plus grande portée, est nettement plus énergivore que le BLE, ce qui le rend moins adapté aux applications IoT de longue durée.

Le LoRa, bien qu'économe en énergie par rapport au Bluetooth classique, utilise plus de puissance que le BLE, notamment en raison des transmissions longue portée, mais reste néanmoins adapté aux applications nécessitant des communications à longue distance avec des débits de données relativement faibles.

4.3. Sécurité

Concernant la sécurité, chaque technologie dispose de mécanismes pour protéger les communications. Le Bluetooth classique dispose de mécanismes de sécurité avancés comme le chiffrement, mais il peut être vulnérable à certains types d'attaques, notamment celles basées sur la découverte du périphérique ou l'interception des communications.

Le BLE, plus récent, propose également des protections de niveau similaire, mais ses mécanismes de sécurité sont souvent plus strictement appliqués, en particulier dans les environnements IoT où des dispositifs à faible consommation sont utilisés. Cependant, la faible puissance de signal peut également rendre les dispositifs plus susceptibles aux attaques de type "man-in-the-middle" si une sécurité supplémentaire n'est pas mise en place.

Le LoRa offre une sécurité robuste, avec des options de chiffrement des données au niveau de la couche application. Son faible débit et sa portée permettent également de réduire la probabilité d'interception des données, bien qu'il soit important de noter que la sécurité de LoRa dépend fortement des paramètres de configuration et des clés de chiffrement utilisées.

Conclusion

La réalisation de cette SAE nous a permis de mettre en pratique une variété de compétences techniques et méthodologiques, tout en répondant à des objectifs concrets. Les différentes étapes du projet, allant de la phase d'analyse à la mise en œuvre des solutions, ont permis de renforcer nos connaissances en gestion de projet, ainsi que vos aptitudes à résoudre des problèmes complexes.

Les résultats obtenus témoignent d'une compréhension approfondie des technologies utilisées, telles que le Bluetooth, le BLE ainsi que le LoRa, et de leur application dans des scénarios pratiques. En particulier, l'analyse de la consommation énergétique et l'autonomie des dispositifs ont mis en lumière des aspects critiques à prendre en compte dans des projets IoT

réels, où l'optimisation des ressources est primordiale. Cela reflète une approche méthodique et scientifique dans l'évaluation des performances.

Table des figures

Figure 1: Système à mettre en place et à étudier 3
Figure 2: Gantt initiale....3
Figure 3: Gantt optimum 4
Figure 4: Capteur ky-012 4
Figure 5: ky-013 4
Figure 6: Graphe montrant l'évolution de la résistance en fonction de la température....5
Figure 7: ESP 32 heltec V2 5
Figure 8: Montage théorique du système avec les capteurs et ESP 32 7
Figure 9: Affichage sur Oled pour bluetooth 8
Figure 10: Affichage sur Oled pour BLE 8
Figure 11: Affichage sur Oled pour LoRa 8
Figure 12: Affichage sur Thingspeak pour Bluetooth 8
Figure 13: Affichage sur Thingspeak pour BLE 9
Figure 14: Affichage sur Thingspeak pour LoRa 9
Figure 15: Plan pour RSSI dans les champs....10
Figure 16: Plan pour RSSI en ville....10
Figure 17: Graphe montrant RSSI en fonction de la distance pour le Bluetooth 11
Figure 18: Graphe montrant RSSI en fonction de la distance pour le BLE 12
Figure 19: Graphe montrant RSSI en fonction de la distance pour le Lora en intérieur 13
Figure 20: Consommation d'énergie du LoRa sur les SF 7 à SF 12 16
Figure 21: Consommation d'énergie du Bluetooth et du BLE....17
Figure 22: Consommation d'énergie du Wifi et des Capteurs 17
Figure 23: Autonomie de la communication Bluetooth 18
Figure 24: Autonomie de la communication BLE 19
Figure 25: Autonomie de la communication LoRa sur les SF è à SF 12....19

Sources

- KY-012 Module buzzer actif - SensorKit
- KY-013 Capteur de température CTN - SensorKit

Annexe 1 : Capteurs KY-012 et KY-013

- Capteur KY-012 :

```
int buzzer = 13;
  void setup() {
  pinMode(buzzer, OUTPUT); // Initialisation de la broche de sortie pour le buzzer
}
 // Boucle du programme principal
void loop() {
  // Le buzzer est activé
  digitalWrite(buzzer, HIGH);
  // Mode d'attente pendant 4 secondes
  delay(4000);
  // le buzzer est désactivé
  digitalWrite(buzzer, LOW);
  // Attente de deux secondes supplémentaires
  delay(2000);
}
```

- Capteur KY-012 :

```
#include <math.h>
int ThermistorPin = 4;
int Vo;
float R1 = 10000; // value of R1 on board
float logR2, R2, T;
float c1 = 0.001129148, c2 = 0.000234125, c3 = 0.0000000876741; //steinhart-hart coefficients for thermistor
void setup() {
  Serial.begin(9600);
}
void loop() {
```

```
  Serial.println();
  Serial.println("- - - - - - - - - - ");
    Serial.println("Code avec log(abs(R2))");
  Vo = analogRead(ThermistorPin);
  Serial.println("Valeur analogique Vo : " + String(Vo));
  R2 = R1 * (1023.0 / (float)Vo - 1.0); //calculate resistance on thermistor
  Serial.println("Résistance calculée R2 : " + String(R2));
  logR2 =  log(abs(R2));
  Serial.println("Log de R2 : " + String(logR2));
  T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2)); // temperature in Kelvin
  Serial.println("Température en Kelvin : " + String(T));
  T=(T-273.15)/2; //convert Kelvin to Celcius
    Serial.println("Temperature : " + String(T) + " C°");
  delay(1000);
}
```

Annexe 2 : Code BLE et WiFi

- Serveur :

```
#include <Wire.h>

#include "HT_SSD1306Wire.h"

#include "BLEDevice.h"

#include <ThingSpeak.h>

#include <WiFi.h>

 //Paramètres WiFi

//const char* ssid = "Bbox-3BFDD79C";

//const char* password = "zK@rsndDg!rwhwmW";

const char* ssid = "PlanetCampus - WiFi prive";

const char* password = "KaF3fXaMKf4Hg";

//const char* ssid = "nico2.4";

//const char* password = "nicolas.";

//const char* ssid = "FabriqueConnectee";

//const char* password = "FGBHN1RN6NF";

 // Déclaration de l'écran OLED

static SSD1306Wire display(0x3c, 500000, SDA_OLED, SCL_OLED,
GEOMETRY_128_64, RST_OLED);

 // Paramètres de ThingSpeak

unsigned long myChannelNumber = 2713143;        // Numéro du canal ThingSpeak

const char *myWriteAPIKey = "1SXASJ0M5GIU3KVM";  // Clé API pour écrire sur
ThingSpeak

 WiFiClient client;  // Objet client pour gérer la connexion Wi-Fi

 // UUIDs pour le service et la caractéristique

static BLEUUID serviceUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b");  // UUID
du service

static BLEUUID charUUID("beb5483e-36e1-4688-b7f5-ea07361b26a8");     // UUID
de la caractéristique
```

```
// Variables pour gérer l'état de la connexion BLE
static boolean doConnect = false;                    // Indique si une connexion est à établir
static boolean connected = false;                    // Indique si la connexion est établie
static boolean doScan = false;                       // Indique si un scan est en cours
static BLERemoteCharacteristic *pRemoteCharacteristic;  // Pointeur vers la caractéristique distante
static BLEAdvertisedDevice *myDevice;                // Pointeur vers le périphérique BLE trouvé
static BLEClient *pClient = nullptr;                 // Pointeur vers le client BLE
// Variables pour l'état du buzzer, la température et le RSSI
String buzzerState = "Buzzer Off";  // État initial du buzzer
String tempMessage = "";            // Message de température
int lastRSSI = 0;                   // Dernier RSSI reçu
// Callback pour les notifications
static void notifyCallback(BLERemoteCharacteristic *pBLERemoteCharacteristic, uint8_t *pData, size_t length, bool isNotify) {
  // Conversion des données reçues en une chaîne de caractères
  String receivedData = String((char *)pData);
  Serial.println("Donnée reçu via BLE: " + receivedData);

  // Traitement des données reçues
  if (receivedData.startsWith("Temp:")) {
    tempMessage = receivedData.substring(5);          // Extraire seulement la température
    connectToWiFiAndSendData(tempMessage.toFloat());  // Envoyer les données à ThingSpeak
  } else if (receivedData.startsWith("Buzzer:")) {
    buzzerState = receivedData.substring(7);  // Extraire l'état du buzzer
  }
```

```
}
 // Classe de callback pour le client BLE
class MyClientCallback : public BLEClientCallbacks {
  void onConnect(BLEClient *pclient) {
    connected = true;  // Indiquer que la connexion est établie
    Serial.println("Connecté au serveur BLE");
  }
   void onDisconnect(BLEClient *pclient) {
    connected = false;  // Indiquer que la connexion est perdue
    Serial.println("Déconnecté du serveur BLE.");
  }
};
 // Fonction pour envoyer les données à ThingSpeak
void sendDataToThingSpeak(float temperature) {
  if (WiFi.status() == WL_CONNECTED) {                              // Vérifier si le Wi-Fi
est connecté
    ThingSpeak.setField(1, temperature);                            // Définir le champ 1 avec
la température
    int result = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);  // Envoyer
les données
    if (result == 200) {
      Serial.println("Données envoyées à ThingSpeak avec succès");
    } else {
      Serial.println("Erreur lors de l'envoi des données à ThingSpeak. Code d'erreur :" +
String(result));
    }
  } else {
    Serial.println("WiFi non connecté");
  }
}
```

```
 // Fonction pour se connecter au serveur BLE
bool connectToServer() {
  Serial.print("Connexion au serveur BLE :");
  Serial.println(myDevice->getAddress().toString().c_str());
   // Création du client BLE
  pClient = BLEDevice::createClient();
  pClient->setClientCallbacks(new MyClientCallback());  // Définir les callbacks
   // Tentative de connexion au périphérique BLE
  if (pClient->connect(myDevice)) {
    BLERemoteService *pRemoteService = pClient->getService(serviceUUID);  // 
Obtenir le service distant
    if (pRemoteService == nullptr) {
      Serial.println("Service non trouvé.");
      pClient->disconnect();  // Déconnexion si le service est introuvable
      return false;
    }
    // Obtenir la caractéristique distante
    pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
    if (pRemoteCharacteristic == nullptr) {
      Serial.println("UUID de la caractéristique non trouvée");
      pClient->disconnect();  // Déconnexion si la caractéristique est introuvable
      return false;
    }
    // Enregistrer le callback pour les notifications
    if (pRemoteCharacteristic->canNotify()) {
      pRemoteCharacteristic->registerForNotify(notifyCallback);
    }
     connected = true;  // Indiquer que la connexion est réussie
    return true;
```

```
  }
   Serial.println("Échec de la connexion au serveur BLE");
  return false;
}
// Fonction pour se connecter au Wi-Fi et envoyer les données
void connectToWiFiAndSendData(float temperature) {
  // Connexion au Wi-Fi
  Serial.println("Connexion au WiFi...");
  WiFi.begin(ssid, password);              // Démarrer la connexion Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {  // Attendre la connexion
    delay(1000);
    Serial.println("Connexion au WiFi...");
  }
  Serial.println("Connecté au WiFi");
   ThingSpeak.begin(client);           // Initialiser ThingSpeak
  sendDataToThingSpeak(temperature);  // Envoyer les données
   // Afficher le RSSI Wi-Fi sur l'écran OLED
  int wifiRSSI = WiFi.RSSI();
  Serial.print("Wi-Fi RSSI: ");
  Serial.println(wifiRSSI);
   // Mettre à jour l'écran OLED avec le Wi-Fi RSSI
  display.clear();
  display.drawString(0, 0, "Mode: BLE!");
  display.drawString(0, 10, "--------------------");
  display.drawString(0, 20, "Temp: " + tempMessage);
  display.drawString(0, 30, "Buzzer: " + buzzerState);
  display.drawString(0, 40, "Wi-Fi RSSI: " + String(wifiRSSI));  // Affiche le RSSI Wi-
Fi
  display.drawString(0, 50, "--------------------");
```

```
  display.display();
   WiFi.disconnect(true);  // Déconnexion du Wi-Fi après envoi
  Serial.println("Déconnecté du WiFi");
}
 // Callback pour les appareils BLE
class MyAdvertisedDeviceCallbacks : public BLEAdvertisedDeviceCallbacks {
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    Serial.print("Appareil BLE annoncé trouvé :");
    Serial.println(advertisedDevice.toString().c_str());
    // Vérifier si l'appareil correspond à notre service
    if (advertisedDevice.haveServiceUUID() &&
advertisedDevice.isAdvertisingService(serviceUUID)) {
      Serial.println("Appareil trouvé ! Connexion...");
      myDevice = new BLEAdvertisedDevice(advertisedDevice);  // Stocker l'appareil
      lastRSSI = advertisedDevice.getRSSI();               // Obtenir le RSSI
      doConnect = true;                                  // Indiquer qu'une connexion est à
établir
      BLEDevice::getScan()->stop();                      // Arrêter le scan
    }
  }
};
 void setup() {
  Serial.begin(115200);  // Initialiser la communication série
  display.init();        // Initialiser l'écran OLED
  display.clear();
  display.display();
   BLEDevice::init("");  // Initialiser le périphérique BLE
  BLEDevice::getScan()->setAdvertisedDeviceCallbacks(new
MyAdvertisedDeviceCallbacks());
```

```
  BLEDevice::getScan()->setActiveScan(true);  // Activer le scan BLE
  BLEDevice::getScan()->start(30, false);     // Démarrer le scan pendant 30 secondes
}
void loop() {
  if (doConnect) {
    if (connectToServer()) {
      doConnect = false;  // Réinitialiser après la connexion
    }
  }
   if (!connected) {
    BLEDevice::getScan()->start(30, false);  // Relancer le scan si déconnecté
  } else if (pClient && pClient->isConnected()) {
    lastRSSI = pClient->getRssi();  // Lire le RSSI BLE
    Serial.print("RSSI actuel : ");
    Serial.println(lastRSSI);
     // Mettre à jour l'écran OLED avec le RSSI BLE
    display.clear();
    display.drawString(0, 0, "Mode: BLE!");
    display.drawString(0, 10, "--------------------");
    display.drawString(0, 20, "Temp: " + tempMessage);
    display.drawString(0, 30, "Buzzer: " + buzzerState);
    display.drawString(0, 40, "BLE RSSI: " + String(lastRSSI));  // Affiche le RSSI BLE
    display.drawString(0, 60, "--------------------");
    display.display();
  }
   delay(4000); // Pause de 4 secondes avant la prochaine itération
}
```

- Client :

```
#include <BLEDevice.h>       // Bibliothèque pour gérer les périphérique BLE

#include <BLEUtils.h>        // Bibliothèque pour les utilisataires BLE

#include <BLEServer.h>       // Bibliothèque pour configurer un serveur BLE

#include <math.h>            // Bibliothèque pour fonctions de mathématiques

#include <Wire.h>            // Bibliothèque pour la communication I2C

#include "HT_SSD1306Wire.h" // Bibliothèque pour gérer l'écran OLED SSD1306

// UUIDs pour le service BLE et ses caractéristiques

#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"         // Identifiant unique pour le service BLE

#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"  // Identifiant unique pour une caractéristique BLE

#define BUZZER_PIN 13                                               // Définition de la broche du buzzer

// Configuration du thermistor (capteur de température)

int ThermistorPin = 4;                                              // Broche analogique où le thermistor est connecté

int Vo;                                                             // Variable pour lire la tension du thermistor

float R1 = 10000;                                                   // Résistance de référence

float logR2, R2, T;                                                 // Variables pour les calculs de résistance et de température

float c1 = 0.001129148, c2 = 0.000234125, c3 = 0.0000000876741;  // Coefficients pour l'équation de Steinhart-Hart

// Déclaration de l'écran OLED

static SSD1306Wire display(0x3c, 500000, SDA_OLED, SCL_OLED, GEOMETRY_128_64, RST_OLED);

// Pointeur pour la caractéristique BLE

BLECharacteristic *pCharacteristic;

void setup() {
```

```
  Serial.begin(115200);                // Initialisation de la communication série pour le débogage
  Serial.println("Starting BLE work!");  // Message de confirmation pour le démarrage BLE
  pinMode(BUZZER_PIN, OUTPUT);          // Initialisation de la broche du buzzer
  // Initialisation de l'écran OLED
  display.init();     // Démarrage de l'écran OLED
  display.clear();    // Efface l'écran
  display.display();  // Affiche un écran vide
  // Initialisation du périphérique BLE
  BLEDevice::init("BuzzerClient");                    // Nom du périphérique BLE
  BLEServer *pServer = BLEDevice::createServer();              // Création du serveur BLE
  BLEService *pService = pServer->createService(SERVICE_UUID);  // Création d'un service BLE avec un UUID spécifique
  // Définition d'une caractéristique BLE avec lecture, écriture et notifications
  pCharacteristic = pService->createCharacteristic(
    CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_NOTIFY);
  pCharacteristic->setValue("Buzzer Off");  // Valeur initiale de la caractéristique
  pService->start();                   // Démarrage du service BLE
  // Configuration de la publicité BLE
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);  // Ajout de l'UUID du service à la publicité
  pAdvertising->setScanResponse(true);         // Réponse au scan
  pAdvertising->setMinPreferred(0x06);         // Réglages de la publicité
  pAdvertising->setMinPreferred(0x12);         // Réglages supplémentaires
  BLEDevice::startAdvertising();               // Démarrage de la publicité BLE
```

```
  Serial.println("Characteristic defined! Now you can control the buzzer!");  // Message de confirmation
}
void loop() {
  // Lecture de la tension du thermistor et calcul de la résistance et température
  Vo = analogRead(ThermistorPin);
  R2 = R1 * (1023.0 / (float)Vo - 1.0);                    // Calcul de la résistance du thermistor
  logR2 = log(abs(R2));                                    // Calcul du logarithme de la résistance
  T = (1.0 / (c1 + c2 * logR2 + c3 * logR2 * logR2 * logR2));  // Calcul de la température en Kelvin
  T = (T - 273.15) / 2;                                    // Conversion en Celsius ajustée
  // Activation ou désactivation du buzzer selon la température
  bool buzzerState = false;
  if (T > 17.15) {                 // Seuil de température pour activer le buzzer
    digitalWrite(BUZZER_PIN, HIGH);  // Active le buzzer
    buzzerState = true;
  } else {
    digitalWrite(BUZZER_PIN, LOW);  // Désactive le buzzer
  }
  // Création de messages pour l'affichage OLED et les notifications BLE
  String tempMessage = "Temp: " + String(T) + " C";                    // Message pour la température
  String buzzerMessage = "Buzzer: " + String(buzzerState ? "ON" : "OFF");  // État du buzzer
  // Affichage des messages sur l'écran OLED
  display.clear();
  display.drawString(0, 0, "Mode: BLE!");
  display.drawString(0, 10, "--------------------");
  display.drawString(0, 20, "Temp: " + tempMessage);
```

```
  display.drawString(0, 30, "Buzzer: " + buzzerState);
  display.drawString(0, 50, "--------------------");
  display.display();  // Applique les changements sur l'écran
  pCharacteristic->setValue(tempMessage.c_str());  // Envoie la température via BLE
  pCharacteristic->notify();                       // Notification du message de température
  Serial.println(tempMessage);
  pCharacteristic->setValue(buzzerMessage.c_str());  // Envoie l'état du buzzer via
BLE
  pCharacteristic->notify();                        // Notification du message du buzzer
  Serial.println(buzzerMessage);
  delay(20000);  // Pause de 20 secondes avant la prochaine itération
}
```

Annexe 3 : Code Bluetooth et WiFi

- Maitre :

```
#include <ThingSpeak.h>       // Bibliothèque pour interagir avec ThingSpeak

#include <WiFi.h>             // Bibliothèque pour la connexion Wi-Fi

#include "HT_SSD1306Wire.h"   // Bibliothèque pour l'écran OLED

#include "BluetoothSerial.h"  // Bibliothèque pour la communication Bluetooth classique

// Initialisation de la bibliothèque Bluetooth Serial

BluetoothSerial SerialBT;

// Nom de l'appareil maître et adresse MAC de l'esclave

String myName = "ESP32-BT-Master";                    // Nom du périphérique Bluetooth maître

uint8_t address[6] = { 0x08, 0x3A, 0x8D, 0x95, 0xA5, 0xAA };  // Adresse MAC de l'esclave

// Configuration de l'écran OLED

static SSD1306Wire display(0x3C, 500000, SDA_OLED, SCL_OLED, GEOMETRY_128_64, RST_OLED);  // Pour OLED 128x64

// Variables pour stocker les informations reçues

int rssi = 0;             // Stocke la valeur RSSI

String temperature = "";   // Température reçue via Bluetooth

String buzzerStatus = "";  // Statut du buzzer reçu via Bluetooth

// Paramètres Wi-Fi

const char* ssid = "PlanetCampus - WiFi prive";

const char* password = "KaF3fXaMKf4Hg";

//const char* ssid = "nico2.4";

//const char* password = "nicolas.";

//const char* ssid = "FabriqueConnectee";

//const char* password = "FGBHN1RN6NF";

// Paramètres pour ThingSpeak

unsigned long myChannelNumber = 2780994;        // Numéro de canal ThingSpeak
```

```
const char* myWriteAPIKey = "6EW3ZHFUHWGTZ62R";  // Clé API pour écrire des données sur ThingSpeak

WiFiClient client;

// Fonction setup() : initialisation du système

void setup() {

  Serial.begin(115200);

  // Initialisation de l'écran OLED

  display.init();

  display.clear();

  display.displayOn();

  display.flipScreenVertically();

  display.setFont(ArialMT_Plain_10);

  display.setTextAlignment(TEXT_ALIGN_LEFT);

  display.drawString(0, 0, "Initializing Master...");

  display.display();

  // Initialisation du Bluetooth en mode maître

  SerialBT.begin(myName, true);

  Serial.println("Master device started.");

  display.drawString(0, 10, "BT: Master Mode");

  display.display();

  // Tentative de connexion à l'esclave Bluetooth

  if (SerialBT.connect(address)) {

    Serial.println("Connected Successfully!");

    display.drawString(0, 20, "BT: Connected!");

    display.display();

  } else {

    Serial.println("Failed to connect.");

    display.drawString(0, 20, "BT: Failed!");

    display.display();
```

```
  }
  ThingSpeak.begin(client);  // Initialisation de ThingSpeak
}
// Fonction pour connecter au Wi-Fi, envoyer les données à ThingSpeak, puis se déconnecter
void sendDataToThingSpeak() {
  Serial.println("Connecting to WiFi...");
  WiFi.begin(ssid, password);
  // Attente de la connexion Wi-Fi
  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 10) {
    delay(1000);
    Serial.print(".");
    attempts++;
  }
  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConnected to WiFi");
    // Envoi des données à ThingSpeak
    ThingSpeak.setField(1, temperature.toFloat());
    int responseCode = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
    if (responseCode == 200) {
      Serial.println("Data sent successfully: Temp = " + temperature + " °C, Buzzer = " + buzzerStatus);
    } else {
      Serial.print("Error sending data. HTTP response code: ");
      Serial.println(responseCode);
    }
    WiFi.disconnect();
    Serial.println("WiFi disconnected after sending data.");
```

```
  } else {
    Serial.println("\nFailed to connect to WiFi.");
  }
}
// Fonction loop() : exécution en boucle
void loop() {
  // Vérifie si des données ont été reçues via Bluetooth
  if (SerialBT.available()) {
    String receivedMessage = SerialBT.readString();  // Lecture du message reçu
    Serial.println("Received: " + receivedMessage);
    // Extraction des données : température et statut du buzzer
    int tempIndex = receivedMessage.indexOf("Temp: ");
    int buzzerIndex = receivedMessage.indexOf("Buzzer: ");
    temperature = receivedMessage.substring(tempIndex + 6,
receivedMessage.indexOf(" °C"));
    buzzerStatus = receivedMessage.substring(buzzerIndex + 8);
    // Affichage des données sur l'écran OLED
    display.clear();
    display.drawString(0, 0, "Mode Bluetooh!");
    display.drawString(0, 10, "Temp: " + temperature + " °C");
    display.drawString(0, 20, "Buzzer: " + buzzerStatus);
    display.drawString(0, 30, "RSSI: " + String(rssi) + " dBm");
    display.display();
    // Envoi des données à ThingSpeak
    sendDataToThingSpeak();
  }
  delay(1000);  // Délai entre chaque itération
}
```

- Esclave :

```
#include <math.h>

#include "HT_SSD1306Wire.h"

#include "BluetoothSerial.h"  // Inclure la bibliothèque Bluetooth Serial

BluetoothSerial SerialBT;

String myName = "ESP32-BT-Slave";

const int buzzer = 13;          // Broche du buzzer

int ThermistorPin = 4;          // Broche du capteur de température

// Initialisation de l'OLED

#ifdef WIRELESS_STICK_V3

static SSD1306Wire display(0x3C, 500000, SDA_OLED, SCL_OLED,
GEOMETRY_64_32, RST_OLED);

#else

static SSD1306Wire display(0x3C, 500000, SDA_OLED, SCL_OLED,
GEOMETRY_128_64, RST_OLED); // Correction ici

#endif

// Variables pour le capteur de température

int Vo;

float R1 = 10000; // Valeur de R1 sur la carte

float logR2, R2, T;

float c1 = 0.001129148, c2 = 0.000234125, c3 = 0.0000000876741;

void setup() {

  Serial.begin(115200);

  SerialBT.begin(myName);

  Serial.printf("L'appareil \"%s\" a démarré en mode esclave, en attente de
connexions...\n", myName.c_str());

  pinMode(buzzer, OUTPUT);

  display.init();

  display.flipScreenVertically();
```

```
  display.setFont(ArialMT_Plain_10);
  display.clear();
}
void loop() {
  Serial.println();
  Serial.println("- - - - - - - - - - ");
  Serial.println("Code avec log(-R2)");
  Vo = analogRead(ThermistorPin);
  Serial.println("Valeur analogique Vo : " + String(Vo));
  R2 = R1 * (1023.0 / (float)Vo - 1.0); //calculate resistance on thermistor
  Serial.println("Résistance calculée R2 : " + String(R2));
  logR2 =  log(abs(R2));
  Serial.println("Log de R2 : " + String(logR2));
  T = (1.0 / (c1 + c2 * logR2 + c3 * logR2 * logR2 * logR2)); // temperature in Kelvin
  Serial.println("Température en Kelvin : " + String(T));
  T = (T - 273.15)/2; //convert Kelvin to Celsius
  Serial.println("Température : " + String(T) + " C°");
  delay(2000);
  // Activation du buzzer si la température dépasse le seuil
  bool buzzerState = T > 17.5;
  digitalWrite(buzzer, buzzerState ? HIGH : LOW);
  // Affichage des données sur l'écran OLED
  display.clear();
  display.drawString(0, 0, "Temp: " + String(T) + " °C");
  display.drawString(0, 15, "Buzzer: " + String(buzzerState ? "ON" : "OFF"));
  display.display();
  // Envoi des données via Bluetooth
  SerialBT.println("Temp: " + String(T) + " °C");
```

```
  SerialBT.println("Buzzer: " + String(buzzerState ? "ON" : "OFF"));
  // Attendre avant la prochaine mise à jour
  delay(20000);
}
```

Annexe 4 : Code LoRa et WiFi

- Emetteur :

```
#include <SPI.h>

#include <LoRa.h>

#include <math.h>

#define SCK 5

#define MISO 19

#define MOSI 27

#define SS 18

#define RST 14

#define DI0 26

#define LED_PIN LED_BUILTIN  // GPIO pour la LED interne

#define freq 899E6

#define sf 7

#define sb 200E3

int buzzer = 13;

int ThermistorPin = 4;

int Vo;

float R1 = 10000; // Value of R1 on board

float logR2, R2, T;

float c1 = 0.001129148, c2 = 0.000234125, c3 = 0.0000000876741; // Steinhart-Hart coefficients for thermistor

union pack {

  uint8_t frame[16];  // Trame en octets

  float data[4];      // Tableau pour les données

} sdp;

void setup() {

  Serial.begin(9600);
```

```
  pinMode(buzzer, OUTPUT);  // Initialisation de la broche de sortie pour le buzzer
  pinMode(DIO, INPUT);
  pinMode(LED_PIN, OUTPUT);  // Configure la LED interne comme sortie
  SPI.begin(SCK, MISO, MOSI, SS);
  LoRa.setPins(SS, RST, DIO);
  if (!LoRa.begin(freq)) {
    Serial.println("Échec de démarrage LoRa !");
    while (1);  // Boucle infinie si LoRa ne démarre pas
  }
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sb);
}
void loop() {
  Serial.println();
  Serial.println("- - - - - - - - - - ");
  Serial.println("Lecture de la température :");
  // Lecture et calcul de la température
  Vo = analogRead(ThermistorPin);
  Serial.println("Valeur analogique Vo : " + String(Vo));
  R2 = R1 * (1023.0 / (float)Vo - 1.0); //calculate resistance on thermistor
  Serial.println("Résistance calculée R2 : " + String(R2));
  logR2 =  log(-R2);
  Serial.println("Log de R2 : " + String(logR2));
  T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2)); // temperature in Kelvin
  Serial.println("Température en Kelvin : " + String(T));
  T=(T-273.15)/4; //convert Kelvin to Celcius
  Serial.println("Température : " + String(T) + " °C");
  // Gestion du buzzer
```

```
  bool buzzerState = false;
  if (T > 17.5) {  // Seuil de déclenchement du buzzer
    digitalWrite(buzzer, HIGH);
    buzzerState = true;
    Serial.println("Buzzer activé !");
    delay(2000);
    digitalWrite(buzzer, LOW);
  } else {
    digitalWrite(buzzer, LOW);
    Serial.println("Buzzer désactivé.");
    delay(2000);
  }
  // Préparation de la trame à envoyer
  sdp.data[0] = T;            // Température
  sdp.data[1] = buzzerState ? 1.0 : 0.0;  // État du buzzer (1.0 pour activé, 0.0 pour
désactivé)
  Serial.println("Données envoyées via LoRa :");
  Serial.println("Température : " + String(sdp.data[0]) + " °C");
  Serial.println("État du buzzer : " + String(sdp.data[1]));
  // Envoi des données via LoRa
  LoRa.beginPacket();
  LoRa.write(sdp.frame, sizeof(sdp.frame));
  LoRa.endPacket();
  delay(20000);  // Pause avant le prochain envoi
}
```

- Récepteur :

```
#include <SPI.h>

#include <LoRa.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>

#include <WiFi.h>

#include <HTTPClient.h>

// LoRa Configuration

#define SCK 5

#define MISO 19

#define MOSI 27

#define SS 18

#define RST 14

#define DI0 26

#define freq 899E6

#define sf 7

#define sb 200E3

// OLED Configuration

#define SCREEN_WIDTH 128

#define SCREEN_HEIGHT 64

#define OLED_RESET 16

#define OLED_SDA 4

#define OLED_SCL 15

#define ESP_NAME "Receiver"

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

// Paramètres Wi-Fi

const char* ssid = "PlanetCampus - WiFi prive";
```

```
const char* password = "KaF3fXaMKf4Hg";
//const char* ssid = "iPhone de Nicolas";
//const char* password = "nicolas.";
//const char* ssid = "Bbox-3BFDD79C";
//const char* password = "zK@rsndDg!rwhwmW";
//const char* ssid = "FabriqueConnectee";
//const char* password = "FGBHN1RN6NF";
// Paramètres de ThingSpeak
unsigned long myChannelNumber = 2781325;        // Numéro du canal ThingSpeak
const char *myWriteAPIKey = "5YKRK0U74UIKLR1Q";  // Clé API pour écrire sur ThingSpeak
union pack {
  uint8_t frame[16];
  float data[4];
} sdp;
float temperature;
bool buzzerState;
int rssi;
void startOLED() {
  pinMode(OLED_RESET, OUTPUT);
  digitalWrite(OLED_RESET, LOW);
  delay(20);
  digitalWrite(OLED_RESET, HIGH);
  // Initialize OLED
  Wire.begin(OLED_SDA, OLED_SCL);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3c, false, false)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }
```

```
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print(ESP_NAME);
  display.display();
}
void connectWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Connexion au WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnecté au WiFi");
}
void sendDataToThingSpeak(float temperature) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String url = "http://api.thingspeak.com/update?api_key=" + String(myWriteAPIKey) +
"&field1=" + String(temperature);
    http.begin(url);
    int httpResponseCode = http.GET();
    if (httpResponseCode > 0) {
      Serial.println("Donnée envoyée à ThingSpeak avec succès !");
    } else {
      Serial.println("Échec d'envoi : " + String(httpResponseCode));
    }
    http.end();
```

```
  } else {
    Serial.println("WiFi non connecté, impossible d'envoyer les données.");
  }
  // Déconnecter du Wi-Fi
  WiFi.disconnect();
  WiFi.mode(WIFI_OFF);
  Serial.println("WiFi déconnecté.");
}
void setup() {
  Serial.begin(9600);
  startOLED();
  SPI.begin(SCK, MISO, MOSI, SS);
  LoRa.setPins(SS, RST, DI0);
  Serial.println("Starting LoRa Receiver!");
  if (!LoRa.begin(freq)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sb);
}
void loop() {
  int packetLen = LoRa.parsePacket();
  if (packetLen == 16) {
    int i = 0;
    while (LoRa.available()) {
      sdp.frame[i] = LoRa.read();
      i++;
```

```
    }
    // Decode received data
    temperature = sdp.data[0];        // Temperature value
    buzzerState = static_cast<bool>(sdp.data[1]); // Buzzer state as boolean
    rssi = LoRa.packetRssi();
    // Display data on Serial Monitor
    Serial.println("- - - - - -");
    Serial.println("Temperature: " + String(temperature) + " °C");
    Serial.println("Buzzer State: " + String(buzzerState ? "ON" : "OFF"));
    Serial.println("RSSI: " + String(rssi));
    // Display data on OLED
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("LoRa Receiver");
    display.println("- - - - - - - -");
    display.print("Temp: ");
    display.println(temperature);
    display.print("Buzzer: ");
    display.println(buzzerState ? "ON" : "OFF");
    display.print("RSSI: ");
    display.println(rssi);
    display.display();
    // Connect to Wi-Fi, send data, and disconnect
    connectWiFi();
    sendDataToThingSpeak(temperature);
  }
}
```

Annexe 5 : Consommation énergétique BLE

- Emetteur

➢ Connexion

Pour calculer la consommation énergétique du BLE lors de la connexion, nous avons observé la variation du signal au moment où le BLE établit une connexion. Les valeurs crête à crête de la connexion (représentées en rouge) ont été mesurées à l'aide d'un oscilloscope. Ensuite, nous avons appliqué la loi d'Ohm : $U = R \times I$, *qui permet de déterminer le courant I à partir de la tension U. La mesure de la tension a été réalisée avec une résistance de 0,5 Ω. Enfin, la consommation énergétique en* W /s a été calculée en multipliant le courant III par la tension d'alimentation selon la formule :

$$Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$$

Données :

- Tension $U_{mesuré}$ = 54,2 mV – 23,2 mV = 31 mV
- Soit I= $\frac{U}{R} = \frac{31\times10^{-3}}{0,5}$= 0,062 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0,062\times3,3}{3600} = 5,68 \times 10^{-5} W/s$

➢ Envoie des données

Pour calculer la consommation énergétique du BLE lors de l'envoi de données, nous avons étudié la variation du signal après l'établissement de la connexion BLE. Les valeurs crête à crête de l'envoi des données (représentées en vert) ont été mesurées à l'aide d'un oscilloscope. Comme pour la connexion, nous avons utilisé la loi d'Ohm : $U = R \times I$, *qui permet de déterminer le courant I à partir de la tension U. La mesure de la tension a été réalisée avec une résistance de*

0,5 Ω. Enfin, la consommation énergétique en W /s a été calculée en multipliant le courant III par la tension d'alimentation selon la formule :

$$Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$$

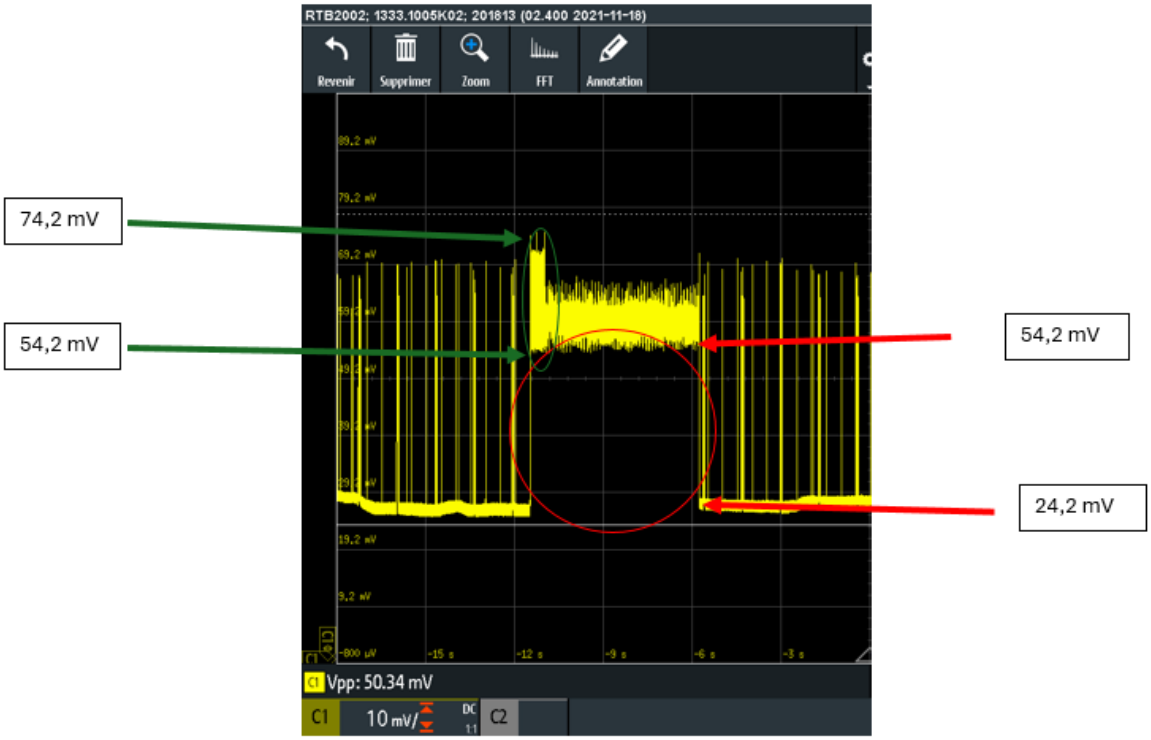
Données :

- Tension $U_{mesuré}$ = 74,2 mV – 54,2 mV = 20 mV
- Soit I= $\frac{U}{R}$ = $\frac{25\times10^{-3}}{0,5}$= 0,04 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0,04\times3,3}{3600}$ = $3,67 \times 10^{-5} W/s$

Ce calcul met en évidence les variations de consommation énergétique entre la connexion et l'envoi des données en BLE.

- Receveur



➢ Connexion

Pour calculer la consommation énergétique du BLE lors de la connexion, j'ai analysé la variation du signal pendant l'établissement de la connexion en BLE. Les valeurs crête à crête de la connexion (représentées en rouge) ont été mesurées à l'aide d'un oscilloscope. Ensuite, la loi d'Ohm ($U = R \times I$) a été utilisée pour déterminer le courant I à partir de la tension U. La tension a été mesurée avec une résistance de 0,5 Ω. Enfin, la consommation énergétique en W/s a été calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{S}\right) = \frac{I\times Tension\ d'alimentation}{Temps}$.

Données :

- Tension $U_{mesuré}$ = 33,34mV
- Soit I= $\frac{U}{R}$ = $\frac{33,34\times10^{-3}}{0,5}$= 0,06668 A
- Consommation énergétique :

Consommation (W /h) :$\frac{0,06668\times3,3}{3600}$ = $6,11 \times 10^{-5} W/s$

➢ Réception des données

Pour calculer la consommation énergétique du BLE à la réception des données, nous avons regardé la variation du signal lorsqu'il recevait les données après connexion en Bluetooth. Puis j'ai pris les valeurs crête à crête de cette réception (représentation en vert). Ensuite, la loi d'Ohm ($U = R \times I$) a été utilisée pour déterminer le courant I à partir de la tension U. La tension a été mesurée avec une résistance de 0,5 Ω. Enfin, la consommation énergétique en W/s a été calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{S}\right) = \frac{I\times Tension\ d'alimentation}{Temps}$.

Données :

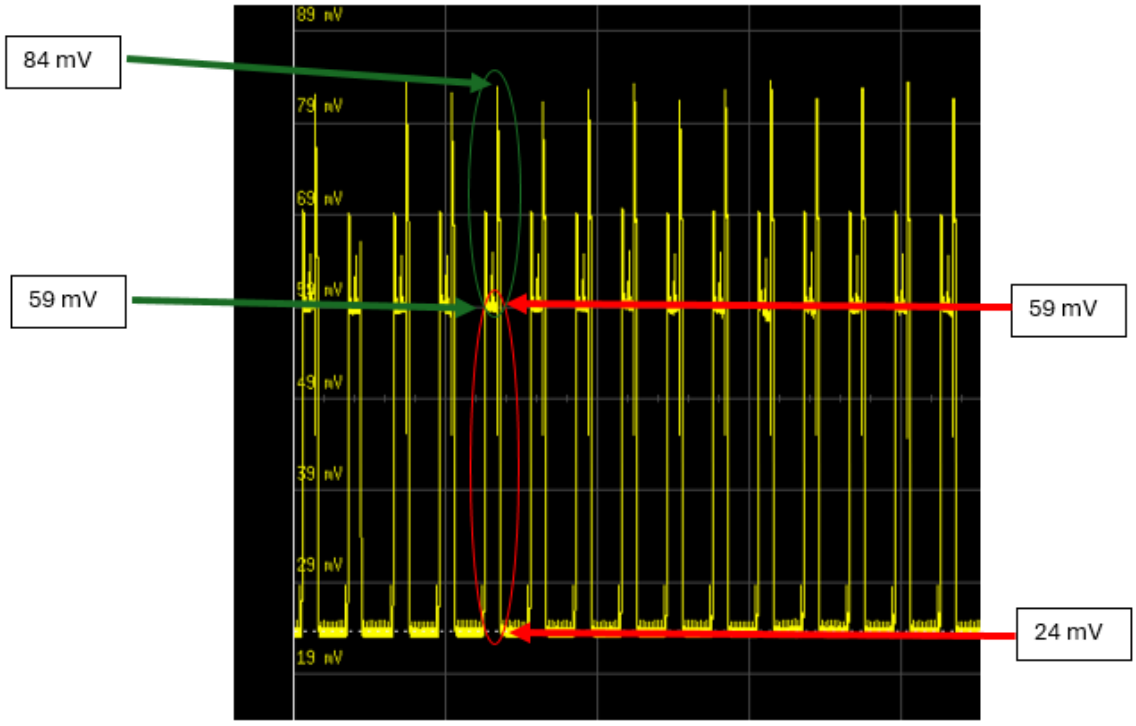
- Tension $U_{mesuré}$ = 74,2 mV – 54,2 mV = 20 mV
- Soit I= $\frac{U}{R}$ = $\frac{25\times10^{-3}}{0,5}$= 0,04 A
- Consommation énergétique :

Consommation (W /s) $\frac{0,04\times3,3}{3600}$ = $3,67 \times 10^{-5} W/s$

Ces calculs permettent de comparer la consommation énergétique entre les étapes de connexion et de réception des données en BLE.

Annexe 6 : Consommation énergétique Bluetooth

- Emetteur



➢ Connexion

Pour calculer la consommation énergétique du Bluetooth à la connexion, nous avons observé la variation du signal pendant cette étape (représentation en rouge). Ensuite, la loi d'Ohm ($U = R \times I$) a été utilisée pour déterminer le courant I à partir de la tension U. La tension a été mesurée avec une résistance de 0,5 Ω. Enfin, la consommation énergétique en W/s a été calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

Données :

- Tension $U_{mesuré}$ = 59 mV - 24 mV = 35 mV
- Soit I= $\frac{U}{R}$ = $\frac{35 \times 10^{-3}}{0,5}$= 0,07 A
- Consommation énergétique :

Consommation (W /h) : $\frac{0,07 \times 3,3}{3600} = 6,42 \times 10^{-5} W/s$

➢ Envoie des données

Pour calculer la consommation énergétique du Bluetooth à l'envoie des données, nous avons regardé la variation du signal lorsqu'il envoyait les données après connexion en Bluetooth. Puis j'ai pris les valeurs crête à crête de cette envoie (représentation en vert). Ensuite, la loi d'Ohm ($U = R \times I$) a été utilisée pour déterminer le courant I à partir de la tension U. La tension a été mesurée avec une résistance de 0,5 Ω. Enfin, la consommation énergétique en W/s a été calculée

en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

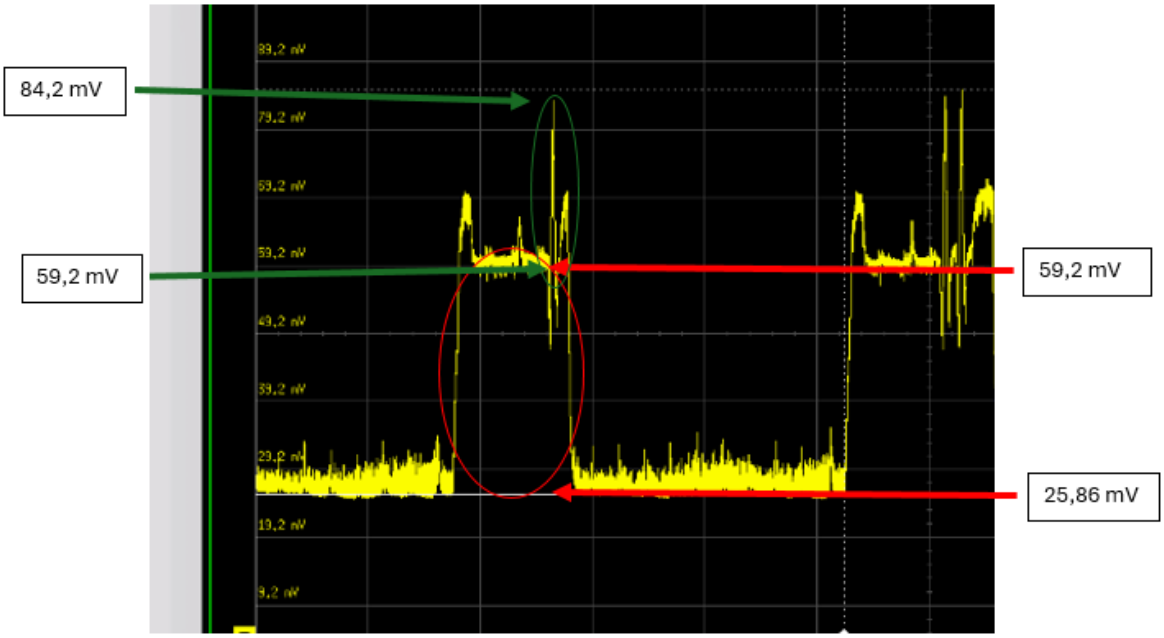
Données :

- Tension $U_{mesuré}$ = 84 mV – 59 mV = 25 mV
- Soit I= $\frac{U}{R}$ = $\frac{25 \times 10^{-3}}{0,5}$= 0,05 A
- Consommation énergétique :

Consommation (W /s) :$\frac{0,05 \times 3,3}{3600} = 4,58 \times 10^{-5} W/s$

Ces résultats montrent une légère différence de consommation énergétique entre les étapes de connexion et d'envoi des données, indiquant que l'envoi consomme légèrement moins que la connexion.

- Receveur



➢ Connexion

Pour calculer la consommation énergétique du Bluetooth à la connexion, nous avons observé la variation du signal pendant cette étape (représentation en rouge). Ensuite, la loi d'Ohm ($U = R \times I$) a été utilisée pour déterminer le courant I à partir de la tension U. La tension a été mesurée avec une résistance de 0,5 Ω. Enfin, la consommation énergétique en W/s a été calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

Données :

- Tension $U_{mesuré}$ = 33,34mV
- Soit I= $\frac{U}{R}$ = $\frac{33,34 \times 10^{-3}}{0,5}$= 0,06668 A

- Consommation énergétique :

$$\text{Consommation (W /h) : } \frac{0{,}06668 \times 3{,}3}{3600} = 6{,}11 \times 10^{-5} W/s$$

➢ Réception des données

Pour calculer la consommation énergétique du Bluetooth lors de la réception des données, nous avons analysé la variation du signal pendant cette phase (représentation en vert). Ensuite, la loi d'Ohm ($U = R \times I$) a été utilisée pour déterminer le courant I à partir de la tension U. La tension a été mesurée avec une résistance de 0,5 Ω. Enfin, la consommation énergétique en W/s a été calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{S}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

Données :

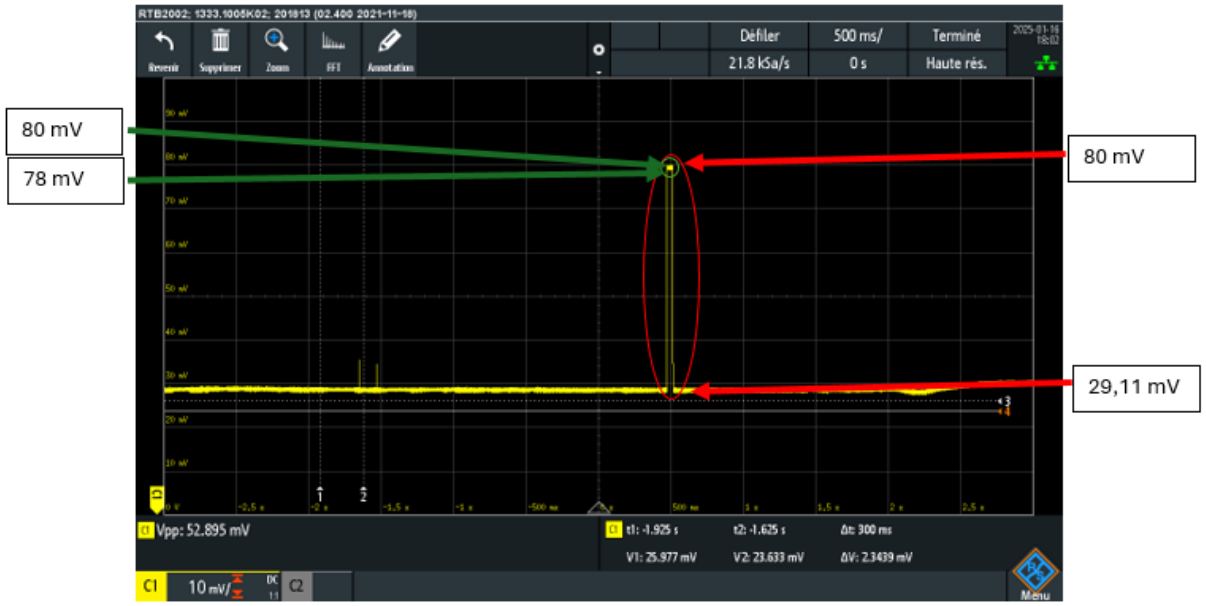
- Tension $U_{mesuré}$ = 25mV
- Soit I= $\frac{U}{R} = \frac{25 \times 10^{-3}}{0{,}5}$= 0,05 A
- Consommation énergétique :

$$\text{Consommation (W /h) : } \frac{0{,}05 \times 3{,}3}{3600} = 4{,}58 \times 10^{-5} W/s$$

Ces résultats montrent que la consommation énergétique du Bluetooth est légèrement plus élevée lors de la connexion que lors de la réception des données.

Annexe 7 : Consommation LoRa

- ❖ Spreading Factor 7 :
- Emetteur :



Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 7 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

- ➢ Connexion :

Données :

- Tension $U_{mesuré}$ =50,895 mV
- Soit I= $\frac{U}{R} = \frac{50{,}895 \times 10^{-3}}{0{,}5}$= 0,10179 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}10179 \times 3{,}3}{3600} = 9{,}33 \times 10^{-5} W/s$

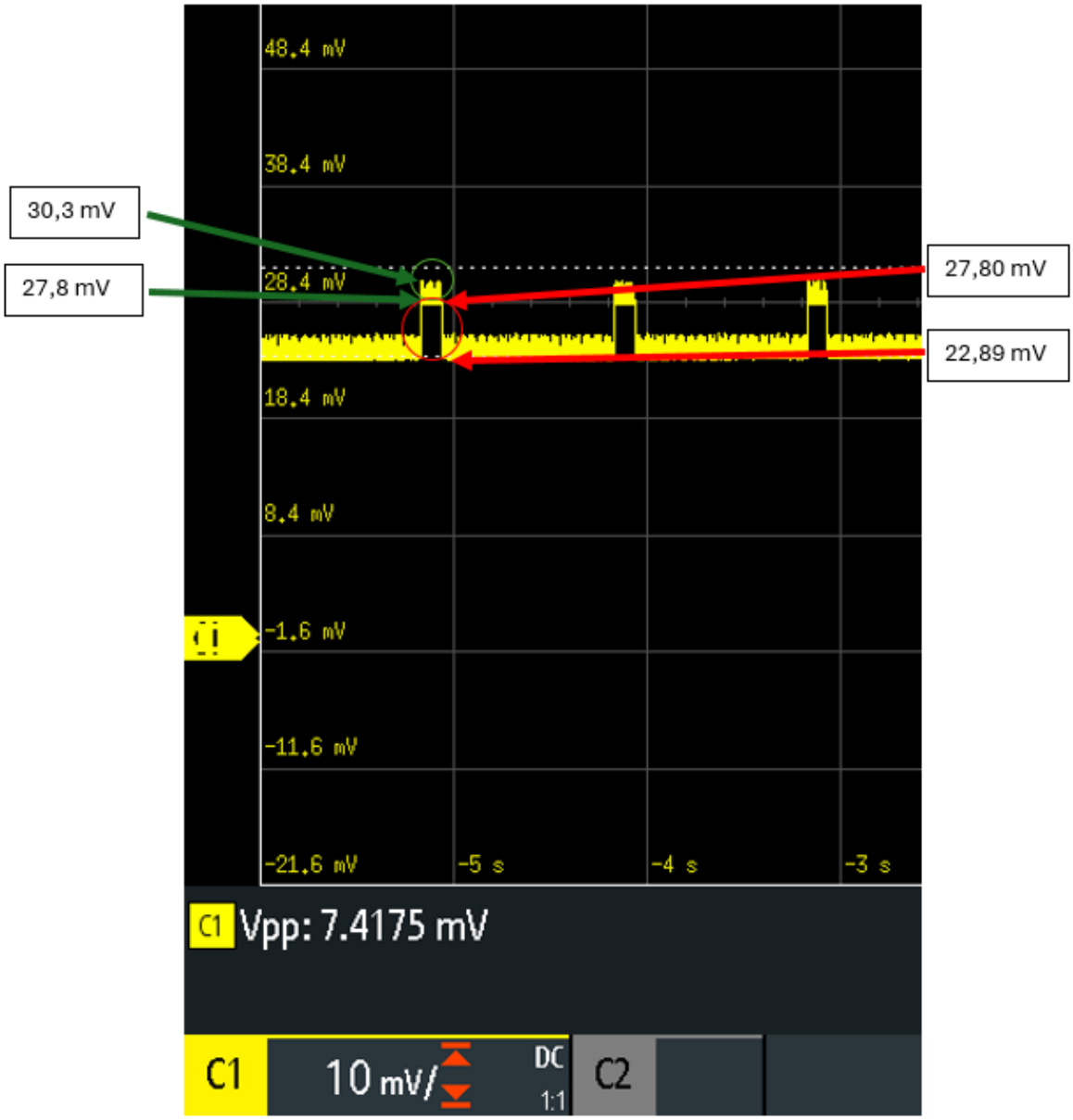
- ➢ Envoie de données :

Données :

- Tension $U_{mesuré}$ =2 mV
- Soit I= $\frac{U}{R} = \frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$

- Receveur



Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 7 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

- Connexion

Données :

- Tension $U_{mesuré}$ = 4,91 mV

- Soit I= $\frac{U}{R} = \frac{4{,}91\times10^{-3}}{0{,}5}$= 0,00982 A
- Consommation énergétique :

$$\text{Consommation (W /s)} : \frac{0{,}00982 \times 3{,}3}{3600} = 9 \times 10^{-6} W/s$$

➢ Réception

Données :

- Tension $U_{mesuré}$ = 2,5 mV
- Soit I= $\frac{U}{R} = \frac{2{,}5 \times10^{-3}}{0{,}5}$= 0,005 A
- Consommation énergétique :

$$\text{Consommation (W /s)} : \frac{0{,}005 \times 3{,}3}{3600} = 4{,}58 \times 10^{-5} W/s$$

❖ Spreading Factor 8 :

Emetteur :

Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 8 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

➢ Connexion :

Données :

- Tension $U_{mesuré}$ = 52,003mV
- Soit I= $\frac{U}{R}$ = $\frac{52{,}003\times10^{-3}}{0{,}5}$= 0,104006 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}104006\times3{,}3}{3600} = 9{,}53\times10^{-5}W/s$

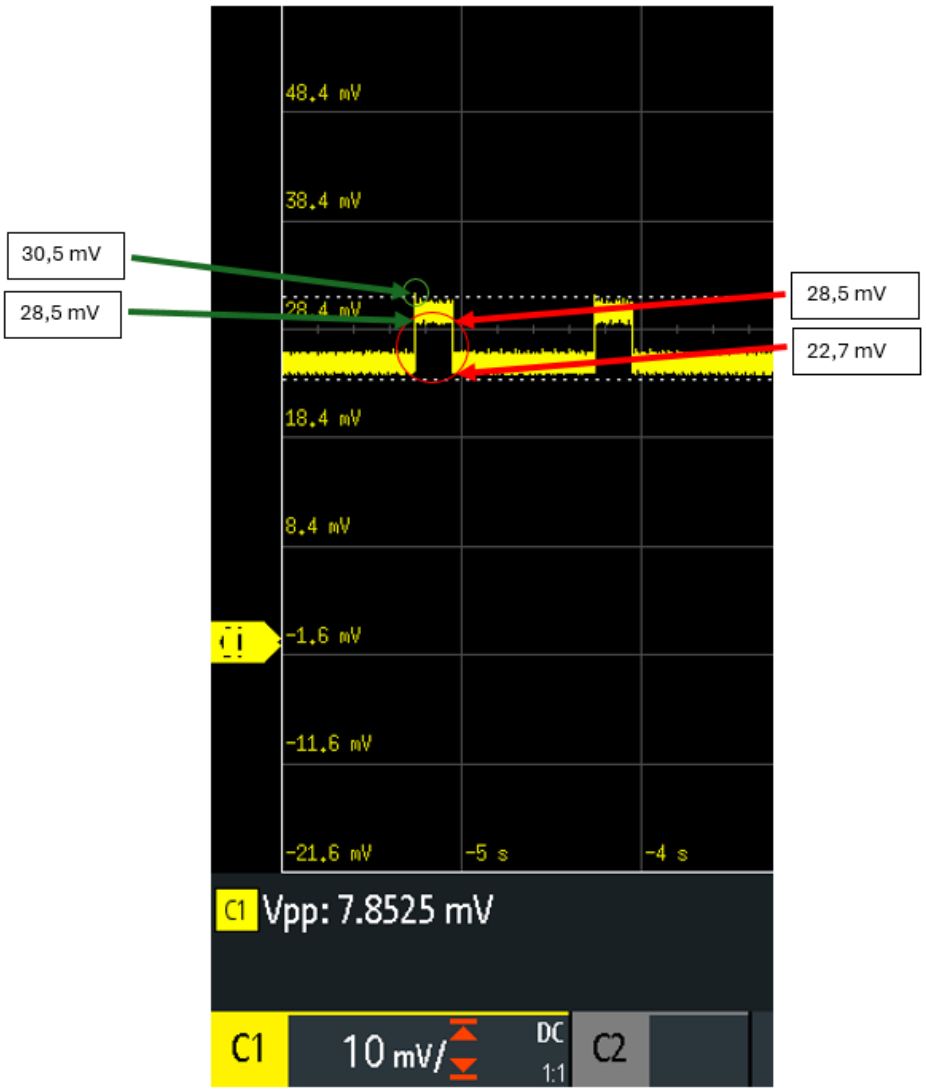
➢ Envoie de données :

Données :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R}$ = $\frac{2\times10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004\times3{,}3}{3600} = 3{,}66\times10^{-6}W/s$

Receveur :



Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 8 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

➢ Connexion

Données :

- Tension $U_{mesuré}$ = 5,8525mV
- Soit I= $\frac{U}{R}$ = $\frac{5{,}8525 \times 10^{-3}}{0{,}5}$= 0,011705 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}011705 \times 3{,}3}{3600} = 1{,}07 \times 10^{-5} W/s$

➢ Réception de données :

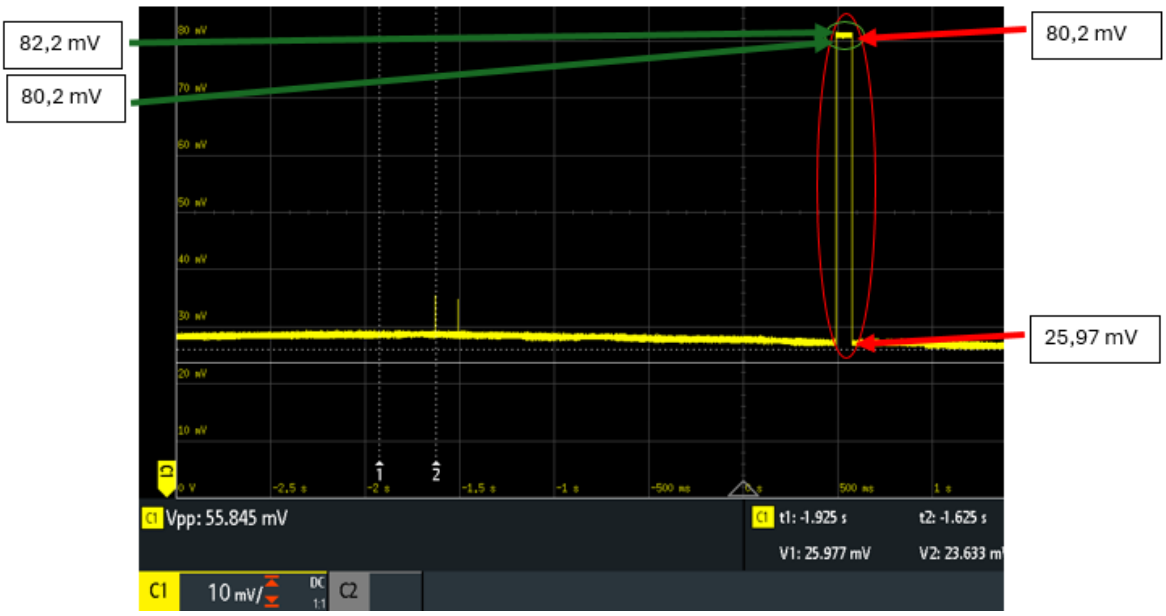
Données :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R}$ = $\frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$

❖ Spreading Factor 9 :

Emetteur :



Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 9 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

➢ Connexion

Données :

- Tension $U_{mesuré}$ = 53,845mV
- Soit I= $\frac{U}{R}$ = $\frac{53{,}845 \times 10^{-3}}{0{,}5}$= 0,10769 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}10769 \times 3{,}3}{3600} = 9{,}87 \times 10^{-5} W/s$

➢ Envoie de données :

Données :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R}$ = $\frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$

Receveur :

Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 9 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

➢ Connexion

Données :

- Tension $U_{mesuré}$ = 5,932mV
- Soit I= $\frac{U}{R}$ = $\frac{5{,}932 \times 10^{-3}}{0{,}5}$= 0,011864 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}011864 \times 3{,}3}{3600} = 1{,}09 \times 10^{-5} W/s$

➢ Reception de données :

Donnée :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R}$ = $\frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$

❖ Spreading Factor 10 :

Emetteur :

Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 10 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

➢ Connexion

Données :

- Tension $U_{mesuré}$ = 59,766 mV
- Soit I= $\frac{U}{R} = \frac{59{,}766 \times 10^{-3}}{0{,}5}$= 0,119532 A
- Consommation énergétique :

$$\text{Consommation (W /s)} : \frac{0{,}119532 \times 3{,}3}{3600} = 1{,}096 \times 10^{-4} W/s$$

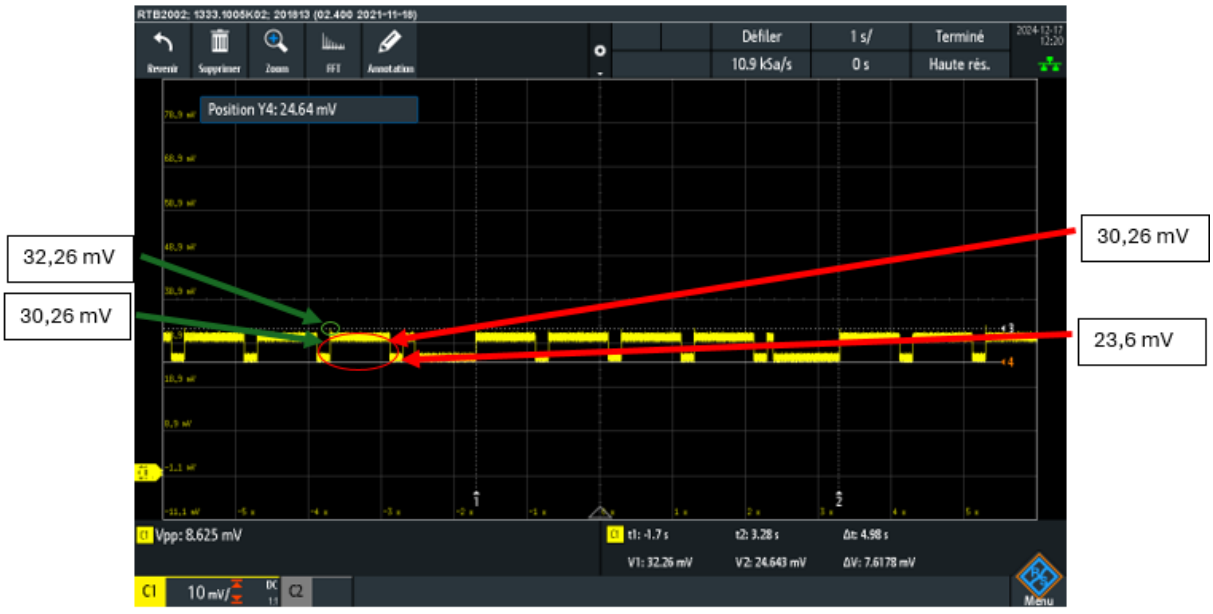
➢ Réception de données :

Données :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R} = \frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

$$\text{Consommation (W /s)} : \frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$$

Receveur :



Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 10 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris

les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

- Connexion

Données :

- Tension $U_{mesuré}$ = 6,625 mV
- Soit I= $\frac{U}{R}$ = $\frac{6{,}625 \times 10^{-3}}{0{,}5}$= 0,01325 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}01325 \times 3{,}3}{3600} = 1{,}2 \times 10^{-5} W/s$

- Reception de données :

Données :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R}$ = $\frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$

- ❖ Spreading Factor 11 :
- Emetteur :

Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 11 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

➢ Connexion

Données :

- Tension $U_{mesuré}$ = 60,271mV
- Soit I= $\frac{U}{R}$ = $\frac{60{,}271 \times 10^{-3}}{0{,}5}$= 0,120542 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}120542 \times 3{,}3}{3600} = 1{,}1 \times 10^{-4} W/s$

➢ Envoie de données :

Données :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R}$ = $\frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$

- Receveur :

Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 11 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

Données :

- Tension $U_{mesuré}$ = 6,579 mV
- Soit I= $\frac{U}{R} = \frac{6{,}579 \times 10^{-3}}{0{,}5}$ = 0,013158 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}013158 \times 3{,}3}{3600} = 1{,}2 \times 10^{-5} W/s$

➢ Réception de données :

Données :

- Tension $U_{mesuré}$ = 6 mV
- Soit I= $\frac{U}{R} = \frac{6 \times 10^{-3}}{0{,}5}$ = 0,012 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}012 \times 3{,}3}{3600} = 1{,}1 \times 10^{-5} W/s$

- ❖ Spreading Factor 12 :
- Emetteur :

Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 12 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

➢ Connexion

Données :

- Tension $U_{mesuré}$ = 62,086mV
- Soit I= $\frac{U}{R} = \frac{64{,}086 \times 10^{-3}}{0{,}5}$= 0, 124172 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}124172 \times 3{,}3}{3600} = 1{,}138 \times 10^{-4} W/s$

➢ Envoie des données

Données :

- Tension $U_{mesuré}$ = 2 mV
- Soit I= $\frac{U}{R} = \frac{2 \times 10^{-3}}{0{,}5}$= 0,004 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}004 \times 3{,}3}{3600} = 3{,}66 \times 10^{-6} W/s$

- Receveur :

Pour calculer la consommation énergétique du LoRa avec le facteur d'étalement 12 à la connexion, nous avons regardé la variation du signal lorsqu'il se connecte en LoRa. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

Données :

- Tension $U_{mesuré}$ = 7,881 mV
- Soit I= $\frac{U}{R} = \frac{7{,}881 \times 10^{-3}}{0{,}5}$ = 0,015762 A
- Consommation énergétique :

$$\text{Consommation (W /s)} : \frac{0{,}015762 \times 3{,}3}{3600} = 1{,}15 \times 10^{-5} W/s$$

➢ Réception de données :

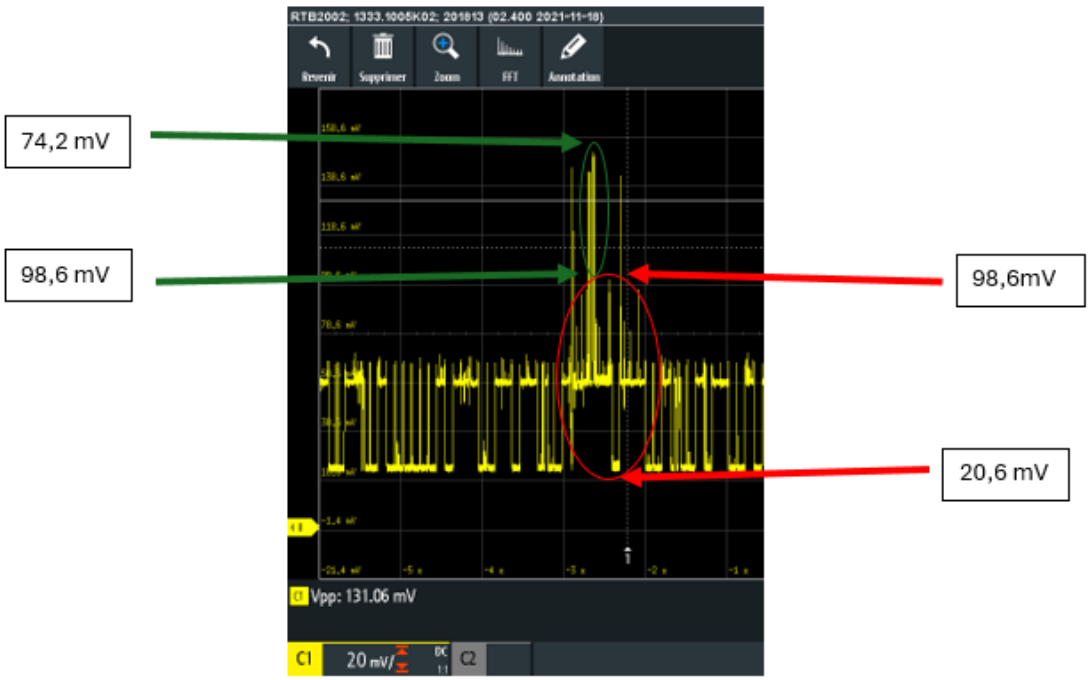
Données :

- Tension $U_{mesuré}$ = 5 mV
- Soit I= $\frac{U}{R} = \frac{5 \times 10^{-3}}{0{,}5}$ = 0,01 A
- Consommation énergétique :

$$\text{Consommation (W /s)} : \frac{0{,}01 \times 3{,}3}{3600} = 9{,}2 \times 10^{-6} W/s$$

Annexe 8 : Consommation énergétique Wifi

- Receveur :



- Connexion

Pour calculer la consommation énergétique du WiFi à la connexion, nous avons la variation du signal lorsqu'il se connecte en WiFi. Puis j'ai pris les valeurs crête à crête de la connexion (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

Données :

- Tension $U_{mesuré}$ = 98,6 mV – 20,6 mV = 78 mV
- Soit I= $\frac{U}{R} = \frac{78 \times 10^{-3}}{0,5}$ = 0,156 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0,0156 \times 3,3}{3600} = 1,43 \times 10^{-5} W/s$

- Envoie des données

Pour calculer la consommation énergétique du WiFI à l'envoie des données, nous avons regardé la variation du signal lorsqu'il envoyait les données après connexion en WiFi. Puis j'ai pris les valeurs crête à crête de cette envoie (représentation en vert). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est

calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\ \left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

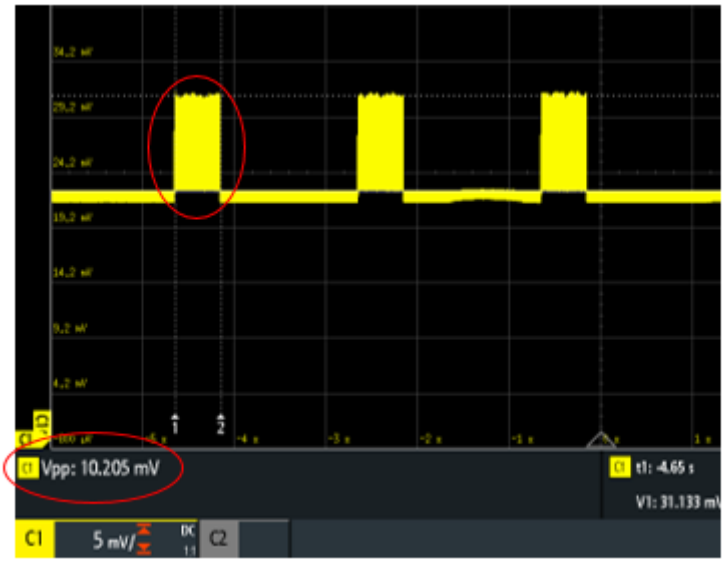
Données :

- Tension $U_{mesuré}$ = 148,6 mV – 98,6mV = 50 mV
- Soit I= $\frac{U}{R}$ = $\frac{50 \times 10^{-3}}{0,5}$= 0,1 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0,1 \times 3,3}{3600} = 9,2 \times 10^{-5} W/s$

Annexe 9 : Consommation énergétique des capteurs

Pour calculer la consommation énergétique du KY-013, nous avons regardé la variation du signal lorsqu'il était utilisé par l'ESP. Puis j'ai pris les valeurs crête à crête de cette envoie (représentation en rouge). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.



Données :

- Tension $U_{mesuré}$ = 10,205 mV
- Soit I= $\frac{U}{R}$ = $\frac{10{,}205 \times 10^{-3}}{0{,}5}$= 0,02041 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0{,}02041 \times 3{,}3}{3600} = 1{,}87 \times 10^{-5} W/s$

➢ KY-013 :

Pour calculer la consommation énergétique du KY-013, nous avons regardé la variation du signal lorsqu'il était utilisé par l'ESP. Puis j'ai pris les valeurs crête à crête de cette envoie (représentation en vert). Puis nous utilisons la loi d'ohm : $U = R \times I$, car elle permet de déterminer le courant I à partir de la tension U, mesurée à l'aide d'un oscilloscope et d'une résistance de 0,5 Ω. Ensuite, la consommation énergétique (W /s) est calculée en multipliant ce courant par la tension d'alimentation, selon la formule : $Consommation\left(\frac{W}{s}\right) = \frac{I \times Tension\ d'alimentation}{Temps}$.

Données :

- Tension $U_{mesuré}$ = 8,5523 mV
- Soit I= $\frac{U}{R}$ = $\frac{8,5523\times10^{-3}}{0,5}$= 0,0171046 A
- Consommation énergétique :

Consommation (W /s) : $\frac{0,0171046\times3,3}{3600}$ = $1,57\times10^{-5}W/s$

Annexe 10 : Fiches solutions

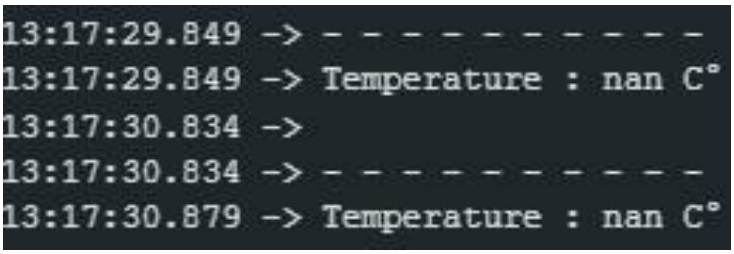
Problème et résolutions liés au capteur ky-013 (temperature sensor)

Introduction

Le capteur KY-013 est un module de mesure de température utilisant un thermistor. Son intégration dans notre projet a posé plusieurs difficultés techniques, notamment en raison des particularités de son thermistor, dont la résistance varie en fonction de la température de manière négative. La compréhension et la résolution de ces problèmes ont nécessité une analyse approfondie du fonctionnement du capteur, des connexions électriques et du code associé.

Problème

Lors de l'utilisation d'un code de base avec le capteur KY-013 sur une carte Heltec, la console affichait « NAN » (Not A Number), indiquant un problème dans la lecture ou le traitement des données. Ce dysfonctionnement empêchait toute exploitation correcte des mesures de température.



Pistes d'investigation

Pour résoudre ce problème, nous avons exploré trois hypothèses principales :

1. Vérifier si le capteur était fonctionnel.
2. Inspecter les connexions et l'intégrité des circuits.
3. Analyser et corriger le code associé au capteur, en identifiant d'éventuelles erreurs de calcul ou de conversion.

Résolution

Vérification des éléments matériels

Nous avons commencé par nous assurer que le capteur KY-013 et ses connexions étaient en bon état de fonctionnement. Une inspection visuelle a permis de confirmer qu'aucun

composant n'était endommagé et que les câbles étaient correctement branchés. Ensuite, nous avons éliminé la possibilité de bruit électrique en stabilisant l'alimentation du capteur.

Pour vérifier si des données étaient transmises, nous avons affiché les valeurs des variables dans le programme. Cette analyse a révélé que des données étaient bien reçues par la variable R2, ce qui a permis d'écarter l'hypothèse d'un capteur ou de connexions défectueuses.

```
15:10:50.259 -> - - - - - - - - - - -
15:10:50.290 -> Code avec log(R2)
15:10:50.290 -> Valeur analogique Vo : 4095
15:10:50.323 -> Résistance calculée R2 : -7501.83
15:10:50.387 -> Log de R2 : nan
15:10:50.387 -> Température en Kelvin : nan
15:10:50.418 -> Temperature : nan C°
15:10:51.367 ->
```

Analyse du code

Nous avons examiné le code responsable du calcul de la résistance à partir de la tension mesurée. La formule utilisée est la suivante :

```
R2 = R1 * (1023.0 / (float)Vo - 1.0); // Calcul de la resistance sur le thermistor

logR2 = log(R2);
```

- **1023** représente la valeur maximale de l'ADC.
- **Vo** est la tension mesurée.
- **R1** est la résistance de référence (10 KΩ dans ce cas).

La valeur calculée pour R2 étant négative, il était impossible de calculer le logarithme népérien, celui-ci étant défini uniquement pour des valeurs strictement positives. Pour éviter cette erreur, nous avons modifié le code afin de prendre la valeur absolue de R2 avant d'appliquer le logarithme :

```
logR2 = log(abs(R2));
```

Correction de la formule de température

Malgré cette correction, la température calculée était de 2 à 4 fois supérieure à la température réelle mesurée avec un thermomètre de référence.

```
13:22:19.714 -> - - - - - - - - - - -
13:22:19.746 -> Code avec log(-R2)
13:22:19.779 -> Valeur analogique Vo : 1213
13:22:19.811 -> Résistance calculée R2 : -1566.36
13:22:19.842 -> Log de R2 : 7.36
13:22:19.842 -> Température en Kelvin : 346.45
13:22:19.875 -> Temperature : 36.65 C°
```

Une analyse plus poussée a révélé une erreur dans la formule de conversion entre la résistance et la température en degrés Celsius. Pour corriger cette surestimation, nous avons ajusté l'échelle avec une division par deux :

T = (T - 273.15) / 2 ; **// Division par 2 pour corriger l'échelle**

Cette modification a permis d'aligner les valeurs calculées sur les résultats attendus, tout en assurant une meilleure précision des mesures.

Validation finale

Après correction, les températures obtenues correspondaient parfaitement aux valeurs réelles mesurées avec un thermomètre de référence. Les ajustements apportés au code ont permis d'éliminer les erreurs initiales, assurant ainsi une utilisation optimale et fiable du capteur KY-013.

- Code final utilisé pour le capteur KY-013 :

SAE_510_temp_ok.ino

```
#include <math.h>
int ThermistorPin = 4;
int Vo;
float R1 = 10000; // value of R1 on board
float logR2, R2, T;
float c1 = 0.001129148, c2 = 0.000234125, c3 = 0.0000000876741; //steinhart-hart coefficients for thermistor
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println();
  Serial.println("- - - - - - - - - - ");
  Serial.println("Code avec log(-R2)");
  Vo = analogRead(ThermistorPin);
  Serial.println("Valeur analogique Vo : " + String(Vo));
  R2 = R1 * (1023.0 / (float)Vo - 1.0); //calculate resistance on thermistor
  Serial.println("Résistance calculée R2 : " + String(R2));
  logR2 =  log(abs(R2));
  Serial.println("Log de R2 : " + String(logR2));
  T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2)); // temperature in Kelvin
  Serial.println("Température en Kelvin : " + String(T));
  T=(T-273.15)/2; //convert Kelvin to Celcius
    Serial.println("Temperature : " + String(T) + " C°");
  delay(1000);
}
```

- Résultat final pour le capteur KY-013 :

```
15:18:04.743 -> - - - - - - - - - - -
15:18:04.779 -> Code avec log(R2)
15:18:04.808 -> Valeur analogique Vo : 1230
15:18:04.839 -> Résistance calculée R2 : -1682.93
15:18:04.871 -> Log de R2 : 7.43
15:18:04.871 -> Température en Kelvin : 344.32
15:18:04.904 -> Temperature : 17.79 C°
15:18:05.857 ->
```

Conclusion

Ce processus de correction a permis de rendre le capteur KY-013 précis et conforme aux exigences de notre projet. En résolvant ces défis techniques, nous avons renforcé notre compréhension des capteurs et des calculs associés. Cette expérience souligne l'importance d'une analyse minutieuse des données, des connexions matérielles et des formules utilisées pour garantir des résultats fiables et exploitables dans les applications pratiques.

Outils et ressources complémentaires

Sites pour le programme du KY-013 :

a. KY-013 Analog Temperature Sensor Module - ArduinoModulesInfo

Documentations techniques :

a. Thermistance NTC | Types de résistances | Guide des résistances | PFCONA