



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER



Master 1 EEA - SME

UE Réalisations systèmes et microsystèmes

Rapport du bureau d'études

Réalisé par:

CHALLALI Ramzi
MESSAOUDI Sara

Année universitaire : 2023-2024

Table des matières

I. Introduction.....	3
II. Manipulation.....	3
1. Matériel utilisé.....	3
2. Projet N°1: Communication LoRa.....	5
2.1. Matériel utilisé :.....	5
2.2. Schéma de câblage:.....	5
2.3. Configuration et initialisation des pins:.....	6
2.4. Fonctionnement du Module LoRa E5:.....	7
2.5. Piloter un module Grove LoRa-E5 avec des commandes AT.....	7
2.4. Communication directe entre modules avec des commandes AT.....	8
2.5. Station de mesure de température et humidité LoRa + SHT31:.....	10
2. Projet N°2: X-Nucleo IKS01A3.....	14
2.1. Caractéristique du X-Nucleo IKS01A3:.....	14
III. Conclusion.....	15

I. Introduction

Pour démarrer notre projet d'étude, nous avons débuté par des expérimentations pratiques visant à nous familiariser avec les opérations de base de la carte et du logiciel, ainsi qu'à comprendre le fonctionnement des capteurs.

Ensuite, nous avons entamé le développement de notre projet de station de mesures connectée, mettant en application toutes les connaissances acquises lors de nos premières manipulations pratiques.

Le projet vise principalement à établir une connexion bidirectionnelle entre deux modules LoRa pour la transmission et la réception de données, tout en mettant en place des protocoles de communication efficaces pour garantir une transmission fiable sur de longues distances. En parallèle, il explore les fonctionnalités du Shield X-NUCLEO-IKS01A3, qui comprend des capteurs de température, de pression, de gyromètre et d'accéléromètre. Ces capteurs permettent la collecte de données environnementales et de mouvement, enrichissant ainsi les informations transmises via la connexion LoRa. En combinant les modules LoRa pour la communication sans fil avec l'intégration des capteurs du Shield X-NUCLEO-IKS01A3, le projet vise à présenter une application pratique et complète de la technologie LoRa dans la collecte et la transmission en temps réel de données environnementales.

II. Manipulation

1. Matériel utilisé

- Carte Nucleo STM32- L476RG:**

Cette carte est dotée d'une capacité de stockage interne permettant à l'utilisateur d'y installer un programme informatique. Ce programme consiste en une série d'instructions destinées à être exécutées par le microcontrôleur qui équipe la carte, un STM32. Elle est idéale pour les applications embarquées en raison de sa faible consommation d'énergie et de ses multiples fonctionnalités. En plus de sa mémoire interne, la carte Nucleo STM32L476RG est pourvue d'une gamme de périphériques intégrés incluant des interfaces de communication et des convertisseurs analogiques-numériques. Ces caractéristiques offrent aux développeurs la possibilité de concevoir des applications complexes et performantes.

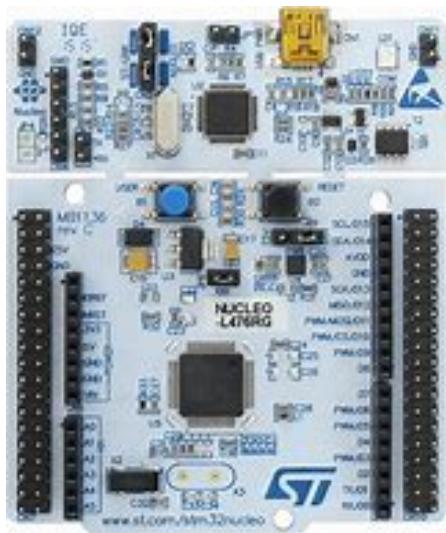


Figure 1: Carte STM32L476RG

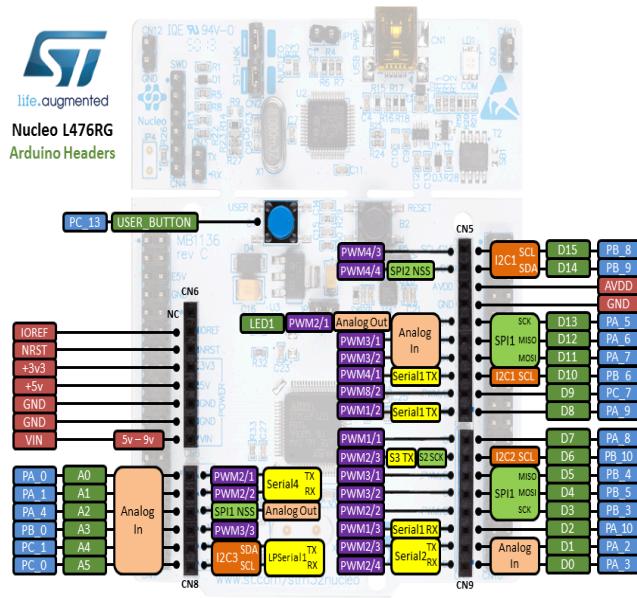


Figure 2: Pinout de la carte

- **Module Grove Base Shield:**

Le module Grove Base Shield simplifie la connexion de modules électroniques aux cartes de développement comme Arduino ou Raspberry Pi. Il utilise des connecteurs standardisés pour les modules Grove, permettant ainsi un prototypage rapide sans câblage compliqué.

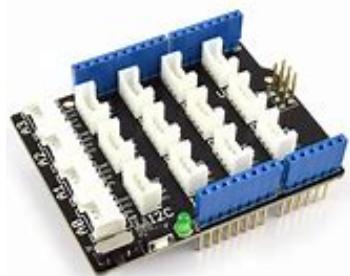


Figure 3: Module Grove Base Shield

- **Module Grove LoRa-E5:**

Le module Grove LoRa-E5 est un dispositif compact conçu pour une communication longue portée à faible consommation d'énergie dans les applications IoT. Compatible avec les normes LoRaWAN, il offre une flexibilité de fréquence, des interfaces Grove standardisées pour une intégration facile, et une faible consommation d'énergie. C'est une solution efficace et pratique pour les projets IoT nécessitant une transmission de données sur de grandes distances.



Figure 4: Module Grove LoRa-E5

2. Projet N°1: Communication LoRa

2.1. Matériel utilisé :

Carte Nucleo STM32L476RG

Shield Base

Module Grove LoRa-E5

Un Picoscope

2.2. Schéma de câblage:

À l'aide du logiciel Fritzing, nous avons élaboré notre schéma de câblage suivant :

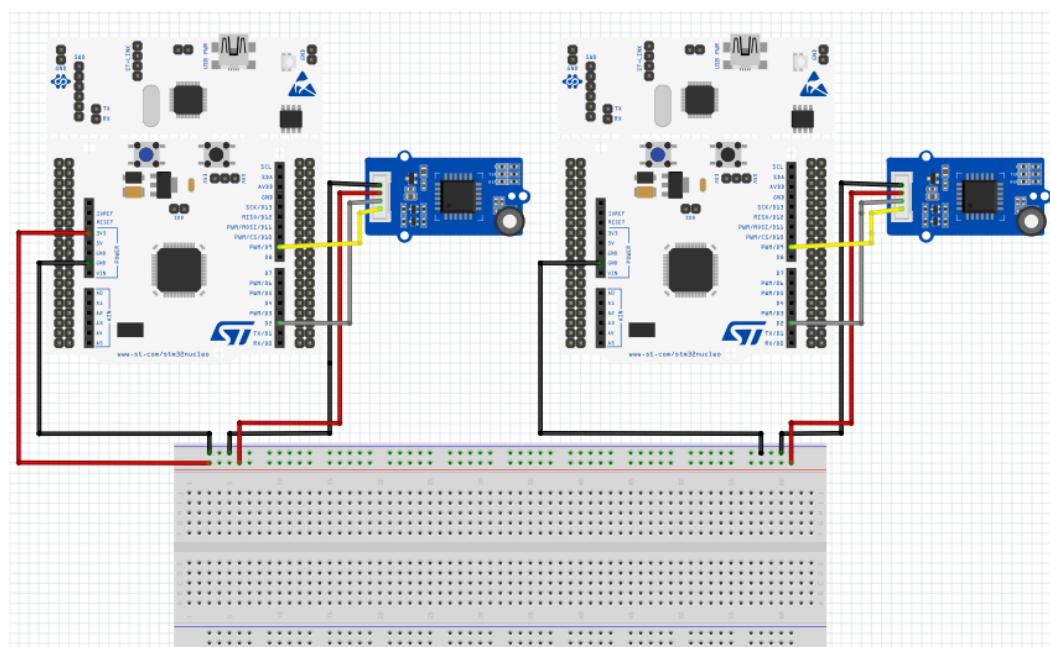


Figure 5: Schéma de câblage de la communication LoRa

2.3. Configuration et initialisation des pins:

La première étape est de s'assurer que les deux modules LoRa sont opérationnels et capables de transmettre des données. Pour ce faire, nous envoyons des commandes de test et attendons leurs réponses, confirmant ainsi leur bon fonctionnement et leur capacité à communiquer entre eux.

Nous avons connecté les pins PA9 et PA10 (UART 1) au USART1_RX et USART1_TX respectivement.

Cette configuration est spécifiquement dédiée à la réception et à la transmission des données du module LoRa. Elle assure un flux bidirectionnel efficace des données entre la carte STM32 et le module LoRa, ce qui est crucial pour la réussite de notre projet.

- baud rate à 9600 bits/s

Cette vitesse de transmission permet d'assurer un transfert efficace des données entre les deux dispositifs, garantissant ainsi une communication fluide et cohérente dans notre application.

- mode asynchrone (Asynchronous)

Pour le mode asynchrone, typiquement utilisé dans les communications série telles que celles avec les capteurs LoRa, les périphériques n'utilisent pas de signal d'horloge commun.

Concrètement, cela signifie que les données sont transmises sans qu'il soit nécessaire d'avoir une synchronisation stricte entre l'émetteur et le récepteur. Chaque octet de données est encadré par un bit de démarrage (start) et suivi d'un ou plusieurs bits d'arrêt (stop), ce qui facilite la synchronisation du récepteur pour la réception des données.

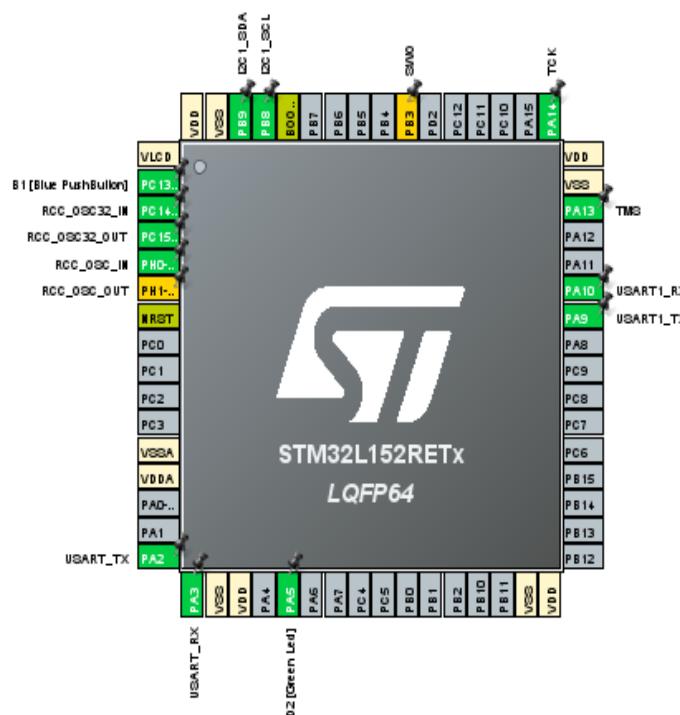


Figure 6: Configuration du STM32

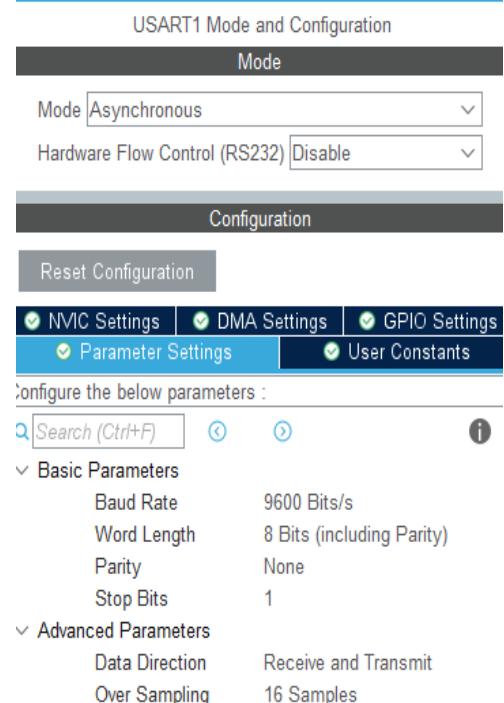


Figure 7: Configuration de l'UART

Remarque: Nous adoptons cette configuration pour tout le reste du projet

2.4. Fonctionnement du Module LoRa E5:

LoRa-E5 est doté d'un STM32WLE5JC haute performance, ce qui convient parfaitement à la conception de divers nœuds IoT. Basé sur les fonctions puissantes et les périphériques riches de STM32WLE5JC, le module fournit UART, I2C, SPI, ADC et GPIO que les utilisateurs peuvent choisir en fonction de l'application. Nous avons besoin de mettre à niveau le micrologiciel de commande AT intégré, nous allons donc utiliser l'interface à deux fils (UART) pour terminer la programmation en fonction du mode de démarrage.

- **Alimentation:** 3,3 et 5 Vcc
- **Consommation en veille:** 60 µA
- **Microcontrôleur:** STM32WLE5JC
- **Microprocesseur:** ARM Cortex-M4 32 bit à 48 MHz
- **Modulation:** (G)FSK, BPSK, (G)MSK et LoRa
- **Interface:** UART
- **Fréquence:** EU 868 Hz et USA 915 Hz
- **Puissance d'émission:** +20 dBm à 3,3 Vcc
- **Sensibilité:** -116,5 dBm à -136 dBm
- **Protocole:** LoRaWAN
- **Portée maxi théorique:** 10 km
- **Antenne:**
 - simple fil (inclus)
 - ou via un connecteur uFL (antenne non incluse)
- **LEDs d'alimentation, Rx et Tx**
- **Température de fonctionnement:** -40 à 85 °C
- **Dimensions:** 40 x 20 mm

Figure 7: Caractéristiques du LoRa E5

2.5. Piloter un module Grove LoRa-E5 avec des commandes AT

Les modules Grove LoRa-E5 se connectent à un port série (UART) et sont pilotés à l'aide de "**commandes AT**". Cela implique que chaque module est doté d'un firmware intégré qui interprète les commandes envoyées sous forme de texte et répond également en texte via le protocole série UART.

Pour cette partie, nous allons assembler deux systèmes identiques. Pour chacun d'eux, on placera la carte d'extension Grove sur la carte NUCLEO et connecter le module LoRa-E5 sur le connecteur UART. (figure 5)

- Test des commandes AT:

Nous avons procédé au test de commandes AT puis AT+ID sur les deux modules LoRa, en se basant sur le code ci-dessous:

```
// Transmet une commande AT au module LoRa
char dataToSend[] = "AT\r"; // Tableau d'octets à envoyer
HAL_UART_Transmit(&huart1, (uint8_t*)dataToSend, strlen(dataToSend), 1000);
// Attente de la réponse à la première commande
HAL_UART_Receive(&huart1, (uint8_t*)receivedData, sizeof(receivedData), 1000);

// Transmission de la réponse reçue à une autre UART (si nécessaire)
HAL_UART_Transmit(&huart2, (uint8_t*)receivedData, strlen(receivedData), 1000);

// Envoi de la deuxième commande "AT+ID"
char secondCommand[] = "AT+ID\r";
HAL_UART_Transmit(&huart1, (uint8_t*)secondCommand, strlen(secondCommand), 1000);

// Attente de la réponse à la deuxième commande (si nécessaire)
HAL_UART_Receive(&huart1, (uint8_t*)receivedData, sizeof(receivedData), 1000);
// Transmission de la réponse reçue à une autre UART (si nécessaire)
HAL_UART_Transmit(&huart2, (uint8_t*)receivedData, strlen(receivedData), 1000);
```

Ce qui donne, sur le moniteur série:

```
+INFO: Input timeout
+AT: OK

+INFO: Input timeout
+ID: DevAddr, 32:30:EB:72
+ID: DevEui, 2C:F7:F1:20:32:30:EB:72
+ID: AppEui, 80:00:00:00:00:00:00:06
```

2.4. Communication directe entre modules avec des commandes AT

A présent, nous pouvons configurer les deux systèmes (chacun constitué d'une carte NUCLEO et d'un module Grove LoRa-E5) pour les faire communiquer. Un sera paramétré comme émetteur, et l'autre comme récepteur.

- Configuration de l'émetteur:

Pour ce faire, nous avons utilisé le code suivant:

```
// Envoi de la 3ème commande "AT+MODE"
char thirdCommand[] = "AT+MODE=TEST\r";
HAL_UART_Transmit(&huart1, (uint8_t*)thirdCommand, strlen(thirdCommand), 1000);

// Attente de la réponse à la 3ième commande (si nécessaire)
HAL_UART_Receive(&huart1, (uint8_t*)receivedData, sizeof(receivedData), 1000);
// Transmission de la réponse reçue à une autre UART (si nécessaire)
HAL_UART_Transmit(&huart2, (uint8_t*)receivedData, strlen(receivedData), 1000);
```

NB: Pour passer en mode test : AT+MODE=TEST

Ce qui donne, sur le moniteur série de l'émetteur :

```
Entrez votre commande
AT+MODE=TEST
Envoyé : AT+MODE=TEST
Message reçu : +MODE: TEST
```

Par défaut, le mode test suppose un comportement émetteur, nous n'avons donc aucune autre commande AT à communiquer au module.

- **Configuration du récepteur:**

Maintenant, on prend l'autre LoRa E5 et on lui adresse les commandes AT suivantes:

```
// Envoi de la 3ème commande "AT+MODE"
char thirdCommand[] = "AT+MODE=TEST\r";
HAL_UART_Transmit(&huart1, (uint8_t*)thirdCommand, strlen(thirdCommand), 1000);

// Attente de la réponse à la 3ième commande (si nécessaire)
HAL_UART_Receive(&huart1, (uint8_t*)receivedData, sizeof(receivedData), 1000);
// Transmission de la réponse reçue à une autre UART (si nécessaire)
HAL_UART_Transmit(&huart2, (uint8_t*)receivedData, strlen(receivedData), 1000);

// Envoi de la 4ème commande "AT+MODE"
char fourthCommand[] = "AT+TEST=RXLRPKT\r";
HAL_UART_Transmit(&huart1, (uint8_t*)fourthCommand, strlen(fourthCommand), 1000);

// Attente de la réponse à la 3ième commande (si nécessaire)
HAL_UART_Receive(&huart1, (uint8_t*)receivedData, sizeof(receivedData), 1000);
// Transmission de la réponse reçue à une autre UART (si nécessaire)
HAL_UART_Transmit(&huart2, (uint8_t*)receivedData, strlen(receivedData), 1000);
```

NB: Pour passer en mode test : AT+MODE=TEST
Pour passer en mode réception : AT+TEST=RXLRPKT

Ce qui donne, sur le moniteur série du récepteur :

```
Entrez votre commande
AT+MODE=TEST
Envoyé : AT+MODE=TEST
Message reçu : +MODE: TEST

Entrez votre commande
AT+TEST=RXLRPKT
Envoyé : AT+TEST=RXLRPKT
Message reçu : +TEST: RXLRPKT
```

- **Envoi d'un message:**

Pour envoyer un message en mode “chaîne de caractères hexadécimaux” :

AT+TEST=TXLRPKT, "0123456789ABCDEF0123456789ABCDEF"

Cette commande ne peut envoyer que des messages constitués des 16 caractères de la base hexadécimale.

Ceci pour envoyer un message en mode “chaîne de caractères ASCII” :

AT+TEST=TXLRSTR, "Bonjour, comment ca va?"

Cette commande peut envoyer n’importe quelle chaîne de caractère ASCII.

Nous n’avons malheureusement pas réussi à faire marcher la commande “**TXLRSTR**”.

Néanmoins, nous avons tout de même utilisé l’autre commande “**TXLRPKT**”, ce qui n’a posé aucun problème car nous n’avons que des données en hexa à transmettre (température et humidité)

```
//Envoi de la commande AT+TEST=TXLRSTR (ascii) pour envoyer un message
//                                         AT+TEST=TXLRPKT (hexa)
char sendCommand[] = "AT+TEST=TXLRPKT AC1234 %s\r";
```

Ce qui donne, sur le moniteur série de l’émetteur :

```
+INFO: Input timeout
+TEST: TXLRPKT "AC1234"
+TEST: TX DONE
```

Et les messages sont bien transmis, on le vérifie sur le moniteur série du récepteur :

```
+INFO: Input timeout
+AT: OK

+INFO: Input timeout
+MODE: TEST

+INFO: Input timeout
+TEST: RXLRPKT
+TEST: LEN:3, RSSI:-41, SNR:13
+TEST: RX "AC1234"
```

2.5. Station de mesure de température et humidité LoRa + SHT31:

Dans la suite de ce projet, notre objectif est désormais d'établir une connexion entre le module LoRa et le capteur SHT31. Nous maintenons le montage du SHT31 et du LoRa avec leurs configurations respectives, et nous les connectons tous les deux à une carte STM32. Notre objectif est de pouvoir lire les données de température et d'humidité du capteur (SHT 31) via le module LoRa.

Pour ce faire, nous devons adapter notre code pour inclure la lecture des données des capteurs, les encapsuler dans un format approprié pour la transmission via LoRa.

Une fois cette étape réalisée, nous serons en mesure de surveiller à distance les conditions de température et d'humidité à partir du capteur via notre réseau LoRa.

- **Schéma de câblage:**

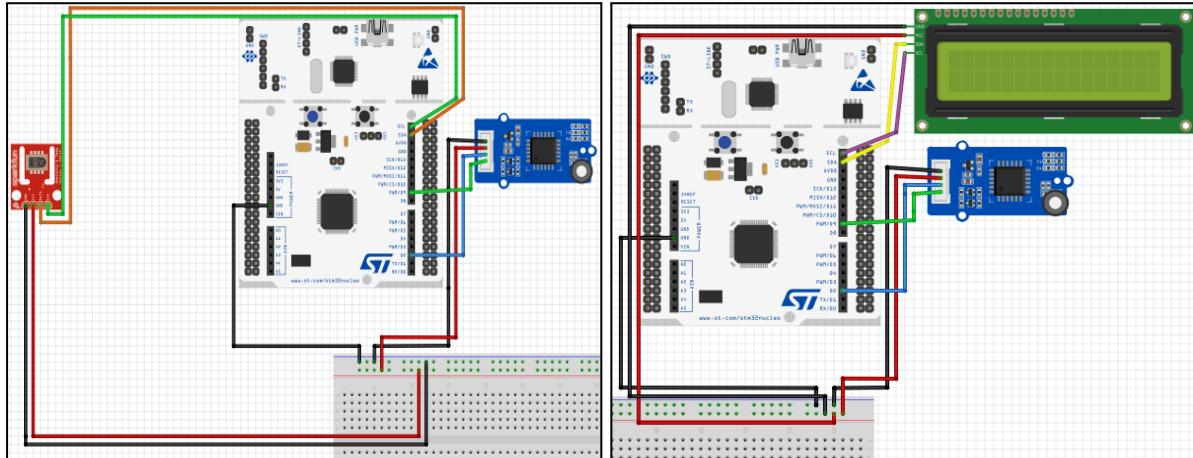


Figure 8: Câblage émission

Figure 9: Câblage réception

- **Les étapes techniques du processus :**

1. Acquisition des données du capteur SHT31 : Nous avons utilisé la carte STM32 pour interagir avec le capteur SHT31 et récupérer les données de température et d'humidité via le protocole de communication I2C.

2. Encapsulation des données pour la transmission LoRa : Une fois les données du capteur SHT31 récupérées, nous les avons encapsulées dans un format adapté au protocole LoRa, optimisant ainsi l'efficacité de la transmission tout en garantissant l'intégrité des données lors de la réception.

3. Transmission des données via LoRa : Le module LoRa a été configuré pour envoyer les données encapsulées en utilisant la modulation LoRa. Nous avons sélectionné une fréquence de transmission et une puissance d'émission appropriées pour assurer une communication fiable et efficace entre les deux modules LoRa.

4. Réception des données par le module LoRa distant : Le deuxième module LoRa a été configuré pour recevoir les données transmises par le premier module LoRa. Les paramètres de réception ont été ajustés pour maximiser la sensibilité du récepteur et améliorer la qualité de réception des données.

5. Affichage des données sur l'écran LCD : Les données de température et d'humidité reçues sont affichées sur un écran LCD connecté à la carte STM32 du module LoRa récepteur. Nous avons utilisé des bibliothèques logicielles adaptées pour contrôler l'affichage des données sur l'écran LCD de manière conviviale.

- Code émission:

Nous avons utilisé une fonction pour convertir les valeurs de température et d'humidité en chaînes de caractères pour les inclure dans une commande à envoyer via LoRa.

Cette fonction prend en entrée la température et l'humidité en tant que flottants, les convertit en chaînes de caractères formatées, puis construit la commande à envoyer en utilisant ces chaînes de caractères.

```
// Conversion de la partie entière de la température en chaîne de caractères
    char temperatureString[10]; // Assez grand pour stocker la température sous forme de chaîne
    char UmidString[10]; // Assez grand pour stocker la humidité sous forme de chaîne
    sprintf(temperatureString, sizeof(temperatureString), "%u%u", (unsigned int)partieEntiere,(unsigned int)partieDecimal);
    sprintf(UmidString, sizeof(temperatureString), "%u",(unsigned int)umid);
    // Construction de la commande à envoyer avec la température
    char sendCommand[30]; // Assez grand pour contenir toute la commande
    sprintf(sendCommand, sizeof(sendCommand), "AT+TEST=TXLRPKT AA%s$AA \r", temperatureString,UmidString);
```

- Code réception:

Pour la réception, nous avons utilisé une fonction pour traiter les données reçues via LoRa. Elle stocke les octets reçus dans un buffer jusqu'à ce qu'elle rencontre un caractère de retour à la ligne (\n). Ensuite, elle analyse la ligne pour extraire les données entre les marqueurs "AA" et "AA", et les convertit en un entier long (long int). Elle assure l'extraction précise des valeurs encapsulées dans les trames LoRa.

```
// Fonction pour traiter les données reçues
long int processReceivedData(char inByte) {
    buffer[index] = inByte;
    index++;
    // Si on reçoit un retour à la ligne, on analyse la ligne
    if (inByte == '\n') {
        // On réinitialise l'index
        index = 0;
        // Si la ligne contient "RX", on la traite
        if (strstr(buffer, "+TEST: RX") != NULL) {
            // On recherche les positions de "AA"
            char *debut = strstr(buffer, "AA");
            char *fin = strstr(debut + 2, "AA");

            if (debut != NULL && fin != NULL) {
                // On extrait les données entre "AA" et "AA"
                char resultat[20];
                strncpy(resultat, debut + 2, fin - debut - 2);
                resultat[fin - debut - 2] = '\0';

                // Conversion de la chaîne de caractères en long int
                extractedValue = strtol(resultat, NULL, 10);
                return extractedValue; } }
        // On efface le buffer
        memset(buffer, 0, BUFFER_SIZE);
    }
    return 0; // Retourne 0 si aucune valeur n'a été extraite
}
```

- **Résultats:**

```
+INFO: Input timeout  
+TEST: TXLRPKT "AA216422AA"  
+TEST: TX DONE  
21.67 C ; 22  
+INFO: Input timeout  
+TEST: TXLRPKT "AA216722AA"  
+TEST: TX DONE  
21.64 C ; 22  
+INFO: Input timeout  
+TEST: TXLRPKT "AA216422AA"  
+TEST: TX DONE  
21.68 C ; 22
```

Figure 10: Résultat émission

```
+INFO: Input timeout  
+TEST: RXLRPKT  
+TEST: LEN:5, RSSI:-27, SNR:13  
+TEST: RX "AA218423AA"  
temperatureint: 21temperaturedec: 78 Humidité: 23  
temperatureint: 21temperaturedec: 82 Humidité: 23  
temperatureint: 21temperaturedec: 84 Humidité: 23  
temperatureint: 21temperaturedec: 78 Humidité: 23  
temperatureint: 21temperaturedec: 85 Humidité: 23  
temperatureint: 21temperaturedec: 78 Humidité: 23  
temperatureint: 21temperaturedec: 77 Humidité: 23  
temperatureint: 21temperaturedec: 80 Humidité: 23
```

Figure 11: Résultat réception

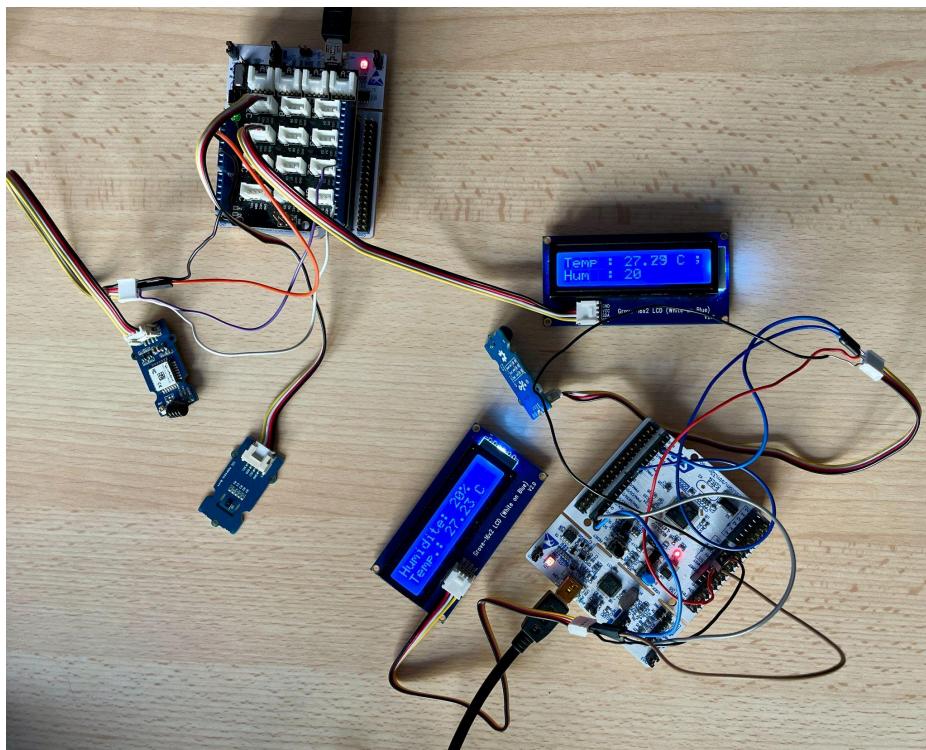


Figure 12: Résultat de la transmission de température et humidité

2. Projet N°2: X-Nucleo IKS01A3

Le X-NUCLEO-IKS01A3 est une carte d'extension de capteurs MEMS et environnementaux pour le STM32 Nucleo. Il est équipé d'une disposition de connecteur Arduino UNO R3 et est conçu autour de l'accéléromètre 3D LSM6DSO et du gyroscope 3D, l'accéléromètre 3D LIS2DW12, le magnétomètre 3D LIS2MDL, le capteur d'humidité et de température HTS221, le LPS22HH capteur de pression et de température et le capteur de température STTS751. Le X-NUCLEO-IKS01A3 s'interface avec le microcontrôleur STM32 ou les cartes Arduino via la broche I²C.

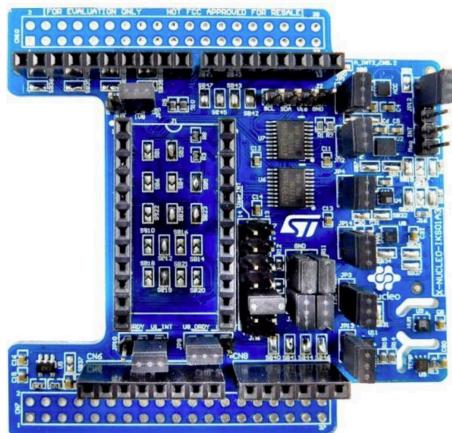


Figure 13: Carte X-Nucleo IKS01A3

2.1. Caractéristique du X-Nucleo IKS01A3:

- LSM6DSO : Accéléromètre MEMS 3D ($\pm 2/\pm 4/\pm 8/\pm 16$ g) + gyroscope 3D ($\pm 125/\pm 250/\pm 500/\pm 1000/\pm 2000$ dps)
- LIS2MDL : Magnétomètre MEMS 3D (± 50 gauss)
- LIS2DW12 : Accéléromètre MEMS 3D ($\pm 2/\pm 4/\pm 8/\pm 16$ g)
- LPS22HH : capteur de pression MEMS, baromètre à sortie numérique absolue 260-1260 hPa
- HTS221 : humidité relative et température numériques capacitives
- STTS751 : Capteur de température (-40 °C à $+125$ °C)
- Prise DIL 24 broches disponible pour des adaptateurs MEMS supplémentaires et d'autres capteurs

Figure 13: Caractéristiques du X-Nucleo IKS01A3

III. Conclusion

Dans l'ensemble, ce projet a été une exploration fascinante des systèmes embarqués, de la communication sans fil et des capteurs environnementaux. Nous avons commencé par des travaux pratiques de base pour nous familiariser avec le matériel, tels que la configuration des cartes STM32, la connexion des capteurs et l'établissement de communications entre les différents composants. Ensuite, nous avons avancé vers des tâches plus complexes, comme l'utilisation des modules LoRa pour instaurer une communication à longue portée entre les capteurs et les microcontrôleurs. Malgré le manque de temps pour intégrer le X-NUCLEO-IKS01A3, nous avons surmonté des défis techniques, comme la gestion des conflits de bus I2C et la sélection des bons paramètres de communication pour garantir une transmission fiable des données. Cette expérience nous a permis de consolider nos connaissances en programmation et d'acquérir des compétences pratiques en manipulant les données des capteurs pour les afficher sur des écrans LCD. En somme, bien que nous n'ayons pas pu exploiter pleinement toutes les ressources disponibles, ce projet nous a offert une expérience enrichissante dans la conception, le développement et le déploiement de systèmes embarqués pour des applications réelles.