



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER



Université
de Toulouse

Partie 2:
Compte Rendu Projet
UE BE

Auteurs : Khemchane ikram - Akel Fadwa

Groupe : M1 SME Groupe AP

Sommaire

1. Introduction.....
2. Matériel utilisé.....
3. But du Projet.....
4. Configuration microcontrôleur.....
4.1 La technologie LORA.....
4.2. Code de test de réponse des 2 modules LoRa.....
4.3. La transmission et l'émission entre les deux Lora.....
4.4. La transmission entre le module Lora et le SHT31et le DHT22.....
5. Résultat LoRa.....
6. X-Nucleo IKS01A3.....
6. Conclusion.....
6.1 Câblage électrique.....
7. Annexes.....

1. Introduction:

Dans le cadre de ce projet d'étude, nous avons initié nos travaux par des expérimentations pratiques afin de nous familiariser avec les opérations fondamentales de la carte et du logiciel, ainsi que le fonctionnement des capteurs. Par la suite, nous avons entrepris le développement du projet de station de mesures connectée, où nous avons appliqué toutes les connaissances acquises lors des travaux pratiques initiaux. Ce projet se concentre particulièrement sur l'intégration et l'utilisation avancée du capteur LoRa en combinaison avec la carte STM32.

2. Matériel utilisé :

- Carte ST NUCLEO-L476RG : Une carte de développement équipée d'un microcontrôleur STM32 de la famille L4, offrant une variété de fonctionnalités et de périphériques pour le développement d'applications embarquées avancées.
- Shield X-NUCLEO-IKS01A3 : Un shield d'extension compatible avec la carte NUCLEO-L476RG, intégrant un capteur de température, de pression, un gyromètre, et un accéléromètre. Il permet la mesure précise de l'environnement et de mouvements.
- Nucleo shield pour L476RG : Un autre shield d'extension conçu spécifiquement pour la carte NUCLEO-L476RG, étendant ses capacités en offrant des connecteurs supplémentaires et des fonctionnalités additionnelles.
- Module Grove LoRa-E5 avec microcontrôleur STM32WLE5JC : Un module de communication LoRa compact, intégrant un microcontrôleur STM32WLE5JC, qui offre des capacités de communication longue portée et faible consommation d'énergie.

- Matrice RGB 8x8 Grove : Une matrice de LED RGB de 8x8 qui offre la possibilité d'afficher des motifs et des animations colorés, permettant une expérience visuelle immersive dans diverses applications.
- Capteur DHT22 et SHT31: Un capteur d'humidité et de température haute précision, qui fournit des mesures fiables dans une large gamme de conditions environnementales.

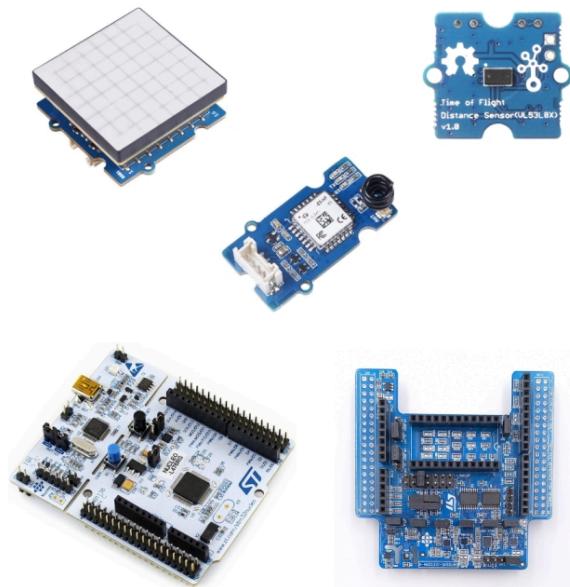


fig.1: Composants dans le IOT sensors kit

3. But Du Projet :

L'objectif principal du projet est de réaliser le fonctionnement simultané de deux modules LoRa en établissant une connexion bidirectionnelle entre eux pour la transmission et la réception de données. Ceci implique la mise en œuvre de protocoles de communication LoRa efficaces pour assurer une transmission fiable et à longue portée des données entre les modules.

En parallèle, le projet vise également à explorer et à utiliser les fonctionnalités offertes par le Shield X-NUCLEO-IKS01A3. Ce shield, équipé de capteurs de température, de pression, de gyromètre et d'accéléromètre, permettra la collecte de données environnementales et de mouvement, enrichissant ainsi les informations transmises via la connexion LoRa.

En combinant l'utilisation des modules LoRa pour la communication sans fil avec l'intégration des capteurs du Shield X-NUCLEO-IKS01A3, le projet vise à démontrer une application pratique et complète de la technologie LoRa

dans le domaine de la collecte et de la transmission de données environnementales en temps réel.

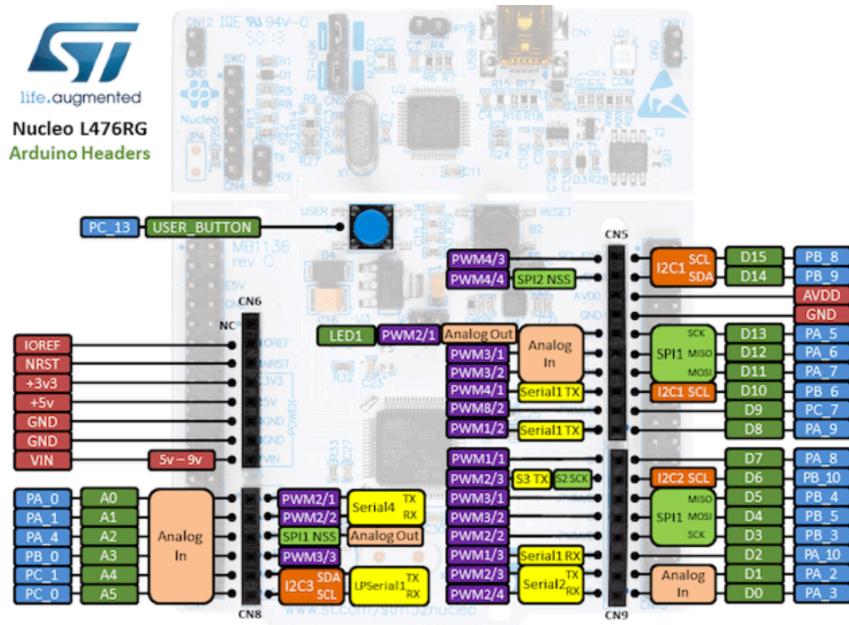


fig.2: Configuration de la carte NUCLEO-L476RG

4. Configuration microcontrôleur :

Après la création du projet sur la carte STM32, la prochaine étape consiste à configurer les broches pour permettre la communication avec les modules LoRa. Cette configuration est réalisée dans l'environnement de développement intégré (IDE), tel que CubeIDE pour STM32, où nous sélectionnons les broches appropriées et définissons leurs fonctions pour la communication avec les modules LoRa.

Une fois la configuration des broches terminée, nous générerons le code en langage C correspondant à cette configuration. Ce code généré est ensuite utilisé pour initialiser les modules LoRa et établir une communication entre eux. La première étape de ce processus est de s'assurer que les deux modules LoRa sont opérationnels et capables de se transmettre des données avec succès. Pour ce faire, nous envoyons des commandes de test aux

modules LoRa et attendons leurs réponses, confirmant ainsi leur bon fonctionnement et leur capacité à communiquer entre eux. Une fois cette étape validée, nous pouvons passer à des tests plus avancés et à l'intégration des modules LoRa dans notre projet global.

Nous avons identifié que les broches PA2 et PA3 étaient configurées pour le USART_RX et le USART_TX, tandis que les broches PA9 et PA10 étaient attribuées au USART1_RX et au USART1_TX respectivement. Après avoir activé ces broches en mode asynchrone (Asynchronous), nous les avons utilisées pour établir une communication avec le module LoRa, permettant ainsi à la carte STM32 de recevoir et de transmettre des données au module LoRa. Il convient de souligner que la configuration des broches PA9 et PA10 pour le USART1_RX et le USART1_TX est spécifiquement dédiée à la réception et à la transmission des données du module LoRa. Cette configuration précise 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. Par ailleurs, nous avons réglé le baud rate à 9600 bits/s, une vitesse de transmission couramment recommandée pour les communications série, afin d'optimiser la fiabilité et la stabilité de la transmission des données. 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.

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. Cette méthode de communication asynchrone est particulièrement adaptée aux applications où une synchronisation stricte n'est pas nécessaire, comme dans les transmissions sans fil à longue portée, telles que celles avec les capteurs LoRa.

La configuration dans le logiciel est la suivante :

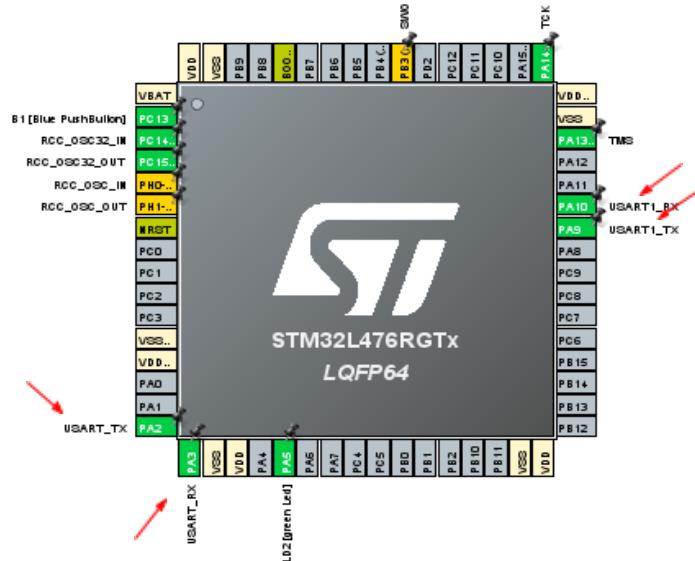


fig.3 : Configuration des pins

4.1. La technologie LORA :

LoRa signifie Long Range Radio et est principalement destiné à l'Internet des objets (IoT) et réseaux. Il s'agit d'une sorte de nouvelle approche de modulation sans fil conçue précisément pour la connectivité longue portée et les communications à faible consommation. Cette technologie permettra aux réseaux multi-locataires ou publics de connecter un certain nombre d'applications fonctionnant sur le même réseau. Et comme certaine datasheet le montre on a un référence design pour la mise en œuvre du LORA :

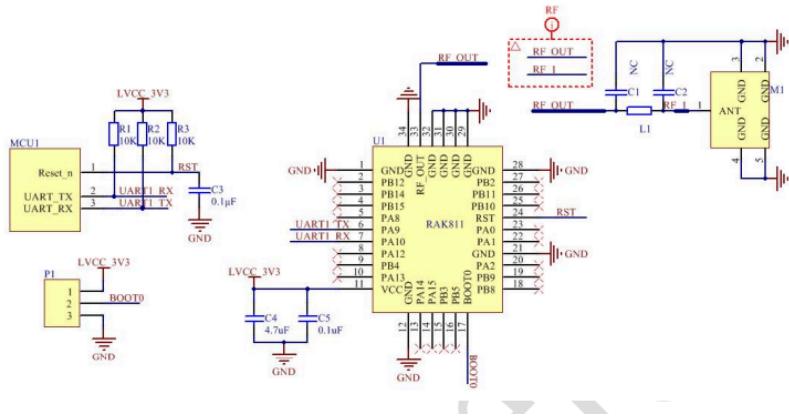


fig.4 : Référence design

Le schéma de câblage pour test de la réponse des 2 modules LORA :

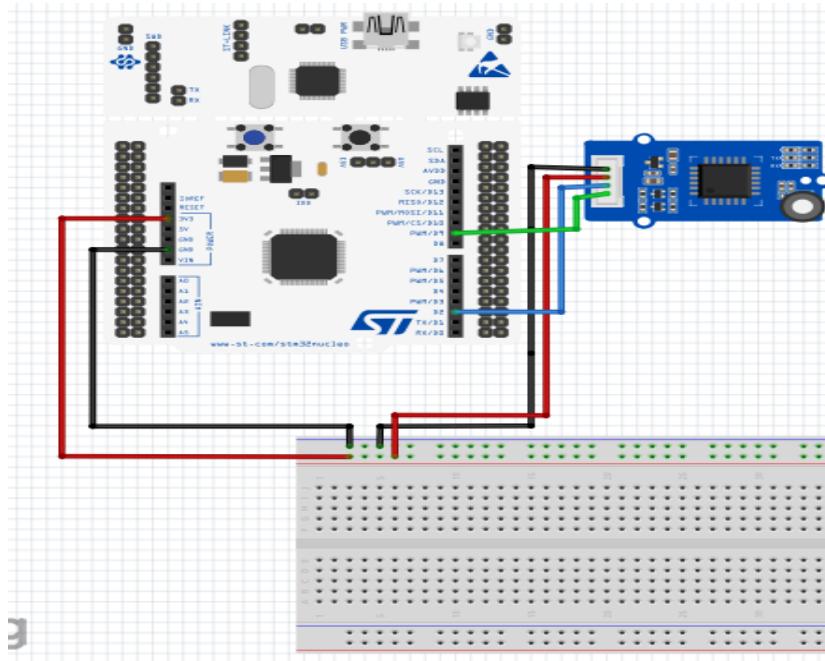


fig.4 : Schéma de câblage du module LoRa

4.2. Code de test de réponse des 2 modules LoRa :

Le code ci-dessous est conçu pour obtenir la réponse de chaque module LoRa en utilisant le langage C sur la carte STM32. Pour afficher la réponse dans un autre logiciel, nous avons utilisé Visual Studio Code (Serial Monitor).

Avec l'ajout de quelques lignes de code en C et le branchement du RX du LoRa avec le TX de la carte STM32 et du RX de la carte dans le TX du LoRa, nous pouvons établir une communication bidirectionnelle entre les deux.

Pour obtenir une réponse de nos modules LoRa, nous envoyons la commande AT\r, ce qui devrait générer une réponse +ok si le module fonctionne correctement et est prêt à communiquer. Ensuite, nous envoyons une deuxième commande, AT+ID\r, pour obtenir l'identificateur du module. Les résultats de ces commandes sont affichés dans le Visual Studio Code (Serial Monitor) pour une vérification visuelle de la communication.

La suite du code et ses détails sont disponibles sur notre compte GitHub pour une référence complète.

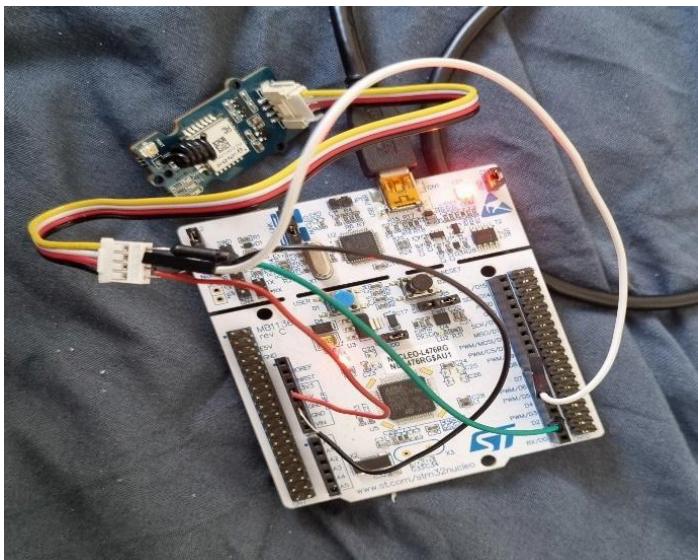


fig.5: Câblage électrique LoRa

Le même câblage et le même code ont été utilisés pour le deuxième module LoRa afin d'obtenir sa réponse. En répétant le processus de configuration et de communication, nous avons pu interroger avec succès le deuxième module LoRa et obtenir sa réponse.

```

char dataToSend[] = "AT\r"; // Tableau d'octets à envoier
//... char dataToSend[] = "AT+ID\r"; la 3ee commande
char receivedData[1000]; // Pour stocker les caractères reçus
/* USER CODE END 0 */

/*
 * @brief The application entry point.
 */
/* Global ATC */
*/
int main(void)
{
    HAL_Init();

    SystemClock_Config();

    MX_GPIO_Init();
    MX_USART2_UART_Init();
    MX_USART1_UART_Init();
/* USER CODE BEGIN 2 */
// Transmit command to LoRa module
    HAL_UART_Transmit(&huart1, (uint8_t*)dataToSend, strlen(dataToSend), 1000);

    // Wait for response from LoRa module
    HAL_UART_Receive(&huart1, (uint8_t*)receivedData, sizeof(receivedData), 1000);

    // Forward the received data to another UART (if needed)
    HAL_UART_Transmit(&huart2, (uint8_t*)receivedData, strlen(receivedData), 1000);
/* USER CODE END 2 */
}

/* USER CODE BEGIN 3 */

```

The terminal window shows the following serial communication:

- +INFO: Input timeout
- +AT: OK
- +INFO: Input timeout
- +ID: DevAddr, 42:00:35:4B
- +ID: DevEui, ZC:F7:F1:20:42:00:35:4B
- +ID: AppEui, 80:00:00:00:00:00:00:06
- +INFO: Input timeout
- +AT: OK
- +INFO: Input timeout
- +ID: DevAddr, 32:30:EB:72
- +ID: DevEui, ZC:F7:F1:20:32:30:EB:72
- +ID: AppEui, 80:00:00:00:00:00:00:06

fig.6: Le code et ça réponse

4.3 La transmission et l'émission entre les deux Lora :

Nous cherchons à établir une communication entre deux modules LoRa, chaque module étant connecté à une carte STM32 L476 RG. Le processus implique l'envoi d'un message par la carte STM32 à l'émetteur LoRa, qui le transmet ensuite au récepteur LoRa. Une fois le message reçu, la carte STM32 récupère les données et affiche la réponse correspondante. Le branchement des broches RX et TX suivra le même principe, garantissant ainsi une communication bidirectionnelle entre les modules LoRa et les cartes STM32.

Le schéma de câblage :

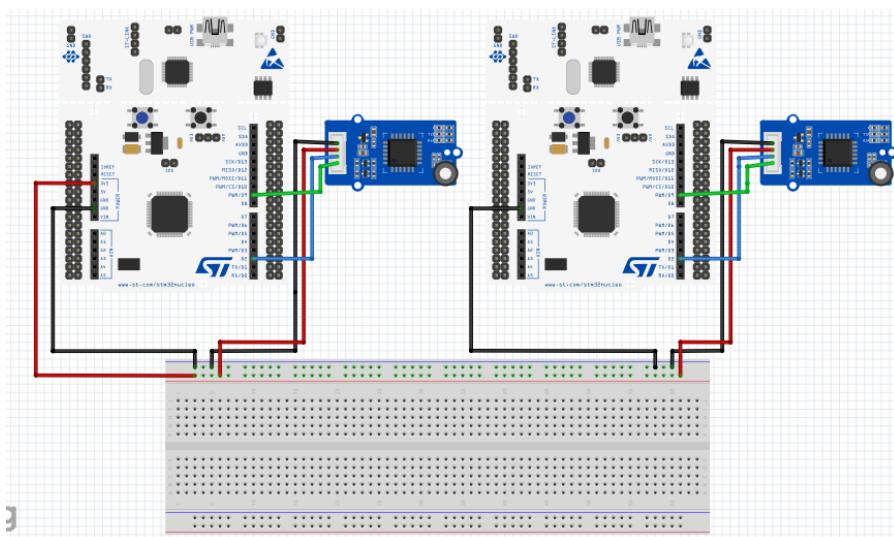


fig.7 : Schéma de câblage de connexion des 2 modules LORA

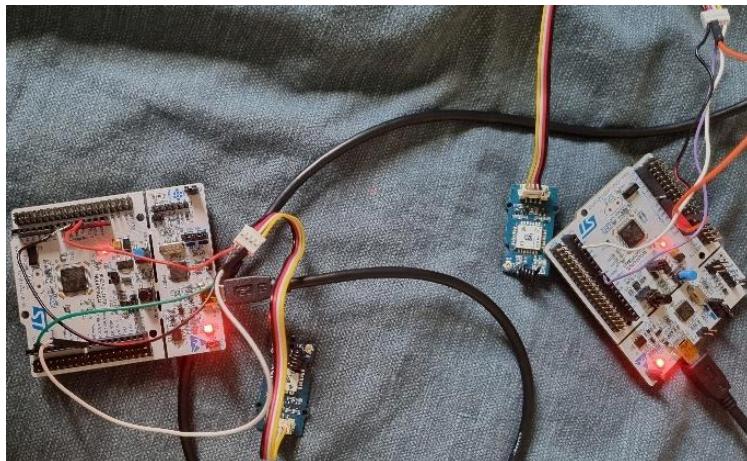


fig.8: La connexion des 2 LoRa

- Le résultat du code de l'émission :

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

fig.9: l'émission du module LoRa

- Le résultat du code de La réception :

```
[  ] +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"  
[  ] +TEST: LEN:3, RSSI:-41, SNR:13  
[  ] +TEST: RX "AC1234"
```

fig.10 : réception du module LoRa

4.4 La transmission entre le module Lora et le SHT31et le DHT22 :

Dans la continuité de ce projet, après avoir confirmé le bon fonctionnement des deux modules LoRa avec une émission et une réception entre eux via

une carte STM32, notre objectif est désormais d'établir une connexion entre le module LoRa et les capteurs SHT31 et DHT22. Nous maintenons le montage du SHT31 et du DHT22 avec leurs configurations respectives, ainsi que le module LoRa, et nous les connectons tous les trois à une carte STM32 et à un Shield de base.

Notre objectif est de pouvoir lire les données de température et d'humidité des deux capteurs via le module LoRa. Pour ce faire, nous devrons adapter notre code pour inclure la lecture des données des capteurs, les encapsuler dans un format approprié pour la transmission via LoRa, et configurer le module LoRa pour la transmission de ces données. Une fois cette étape réalisée, nous serons en mesure de surveiller à distance les conditions de température et d'humidité à partir des deux capteurs via notre réseau LoRa.

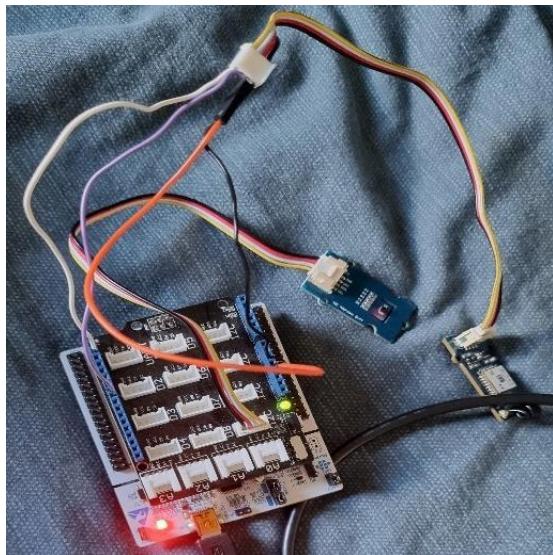


fig.11: Montage module LoRa au capteur SHT31

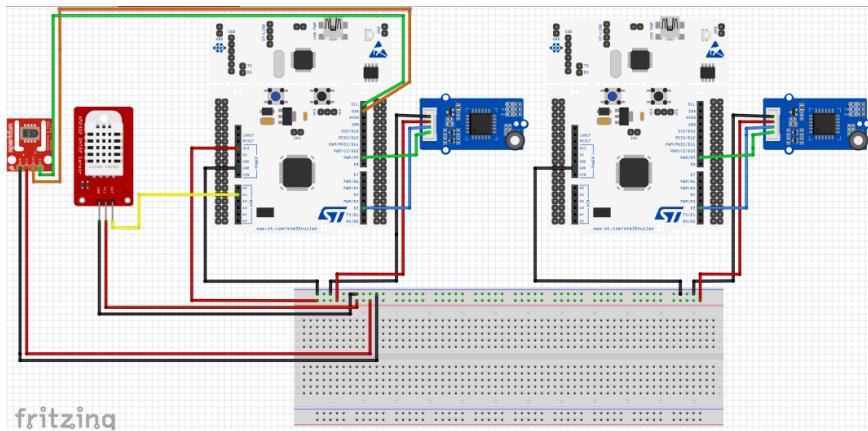


fig.12 : Schéma de câblage LoRa +SHT31+DHT22

La transmission et la réception des données du capteur SHT31 via le module LoRa ont été effectuées avec succès. Voici les détails techniques de notre démarche :

- 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. Le capteur SHT31 utilise un protocole de communication I2C pour transmettre les données à la carte STM32.
- Encapsulation des données pour la transmission LoRa : Après avoir récupéré les données du capteur SHT31, nous les avons encapsulées dans un paquet de données adapté au protocole de communication LoRa. Les données ont été formatées de manière à optimiser l'efficacité de la transmission et à garantir l'intégrité des données lors de la réception.
- Transmission des données via LoRa : Le module LoRa a été configuré pour transmettre les données encapsulées au moyen de la modulation LoRa. Nous avons choisi une fréquence de transmission et une puissance d'émission appropriées pour assurer une communication fiable et efficace entre les deux modules LoRa.
- 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.
- Affichage des données sur l'écran LCD : Les données de température et d'humidité reçues ont été 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 avec une interface simple et conviviale.

5. Résultat LoRa:

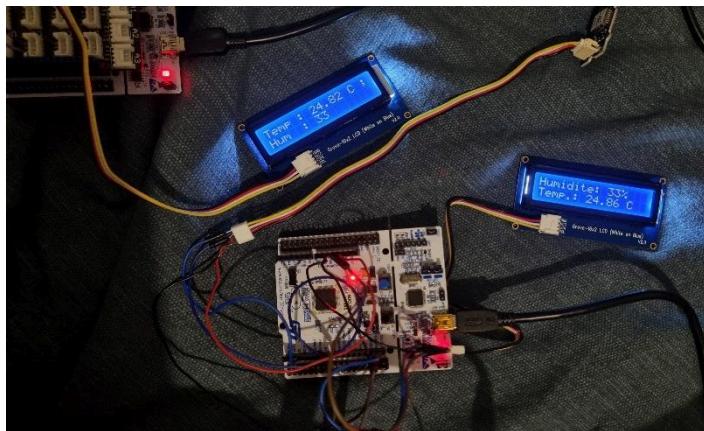


fig.13: Câblage et résultat des 2 lora + SHT31

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

fig.14 : Le résultat de la transmission et la réception du lora et SHT31 .

6. X-Nucleo IKS01A3:

Le X-NUCLEO-IKS01A3 est un système d'évaluation de capteurs de mouvement MEMS et environnementaux. Il est compatible avec la disposition des connecteurs Arduino UNO R3 et comprend les composants suivants : l'accéléromètre + gyroscope 3 axes LSM6DSO, le magnétomètre 3 axes LIS2MDL, l'accéléromètre 3 axes LIS2DW12, le capteur d'humidité et de température HTS221, le capteur de pression LPS22HH et le capteur de température STTS751.

Le X-NUCLEO-IKS01A3 se connecte au microcontrôleur STM32 via la broche I²C, et il est possible de modifier le port I²C par défaut.

- Caractéristiques :

- 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 de 260 à 1260 hPa
- HTS221 : Capteur d'humidité relative et de température numérique capacitive
- STTS751 : Capteur de température (-40 °C à +125 °C)
- Prise DIL 24 broches disponible pour des adaptateurs MEMS supplémentaires et d'autres capteurs
- Bibliothèque de développement complète et exemple gratuits pour tous les capteurs compatibles avec le firmware STM32Cube
- Fonctionnalités de concentrateur de capteurs I²C disponibles sur LSM6DSO
- Compatible avec les cartes Nucleo STM32
- Équipé du connecteur Arduino UNO R3
- Conforme à la directive RoHS
- Conforme à la directive DEEE

Le X-NUCLEO-IKS01A3 offre une solution complète pour l'évaluation des capteurs MEMS et environnementaux, avec une large gamme d'applications potentielles dans les domaines de la surveillance de l'environnement, des dispositifs portables, de l'IoT et bien d'autres encore.

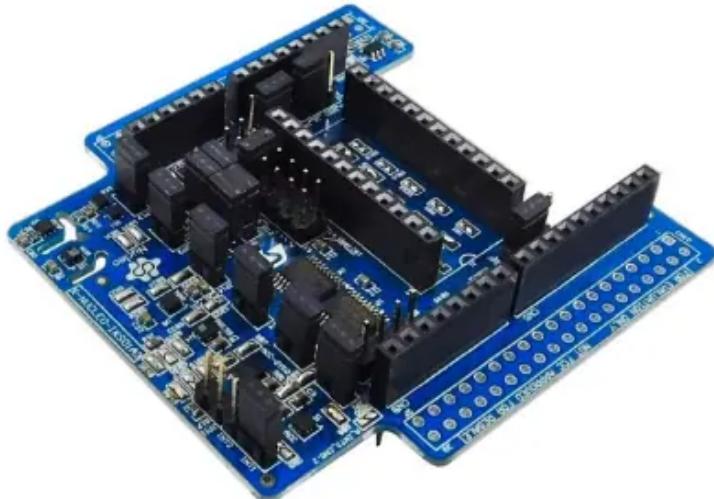


fig.14: X-NUCLEO IKS01A3

6.1 Configuration IKS01A3 dans le STM32CubeIDE:

Pour configurer le composant IKS01A3, nous avons tout d'abord configuré la communication I2C1 sur les broches PB9 (SDA) et PB10 (SCL). L'I2C (Inter-Integrated Circuit) est un protocole de communication série synchrone qui permet la communication entre plusieurs périphériques sur le même bus. En configurant l'I2C, nous permettons à notre microcontrôleur de communiquer avec les capteurs présents sur le X-NUCLEO-IKS01A3, tels que l'accéléromètre, le gyroscope, le magnétomètre, le capteur de température, le capteur d'humidité, le capteur de pression, etc. La configuration de l'I2C1 sur les broches PB9 et PB10 est réalisée en raison de la compatibilité matérielle avec le X-NUCLEO-IKS01A3. Les broches PB9 et PB10 sont les broches dédiées au bus I2C1 sur la plupart des cartes STM32. En configurant ces broches pour l'I2C1, nous assurons une connexion correcte et stable entre le microcontrôleur STM32 et le X-NUCLEO-IKS01A3.

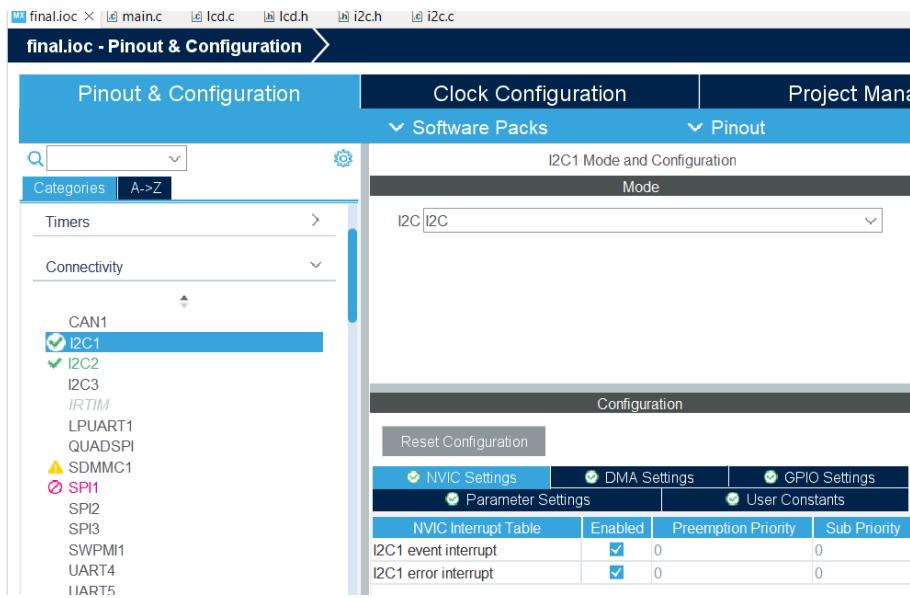


fig. 15: Configuration I2C1 et les interrupteurs

Ensuite, dans la section "Software Packs", nous accédons à "Manage Software Packs" puis sélectionnons "ST Microelectronics". À partir de là, nous choisissons le X-CUBE-MEMS1 et téléchargeons les pilotes pour les composants MEMS.

Le package logiciel d'extension X-CUBE-MEMS1 pour STM32Cube fonctionne sur les microcontrôleurs STM32 et comprend des pilotes qui reconnaissent les capteurs et collectent les données de température, d'humidité, de pression et de mouvement. Cette extension est construite sur la technologie logicielle STM32Cube pour faciliter la portabilité entre différents microcontrôleurs STM32. Le logiciel est accompagné d'une implémentation d'exemple des pilotes s'exécutant sur les cartes d'extension X-NUCLEO-IKS4A1/X-NUCLEO-IKS01A3/X-NUCLEO-IKS02A1 connectées à une carte de développement STM32 Nucleo en vedette. Le logiciel est également disponible sur GitHub, où les utilisateurs peuvent signaler des bugs et proposer de nouvelles idées via les onglets Issues et Pull Requests.

Application		Applications			
Middleware	MotionAC	MotionAD	MotionAR	MotionAT	
	MotionAW	MotionCP	MotionDI	MotionEC	
	MotionFA	MotionFD	MotionFX	MotionGC	
	MotionGR	MotionID	MotionMC	MotionPE	
	MotionPM	MotionPW	MotionSD	MotionSM	
	MotionSP	MotionTL	MotionVC	InfraredPD	
	STM32Cube Hardware Abstraction Layer (HAL)				
Hardware Abstraction	STM32 Nucleo expansion boards X-NUCLEO-IKS01A3 (Sense) X-NUCLEO-IKS02A1 (Sense) X-NUCLEO-IKS4A1 (Sense)				
STM32 Nucleo development board					
Calibration Algorithms	Magnetometer Calibration	Gyroscope Calibration	Accelerometer Calibration	Industrial Algorithms	FFT & Vibration Monitoring
Position Tracking	Sensor Fusion	eCompass	Tilt Sensing	Vertical Context	Airplane Detection
Activity Tracking for Mobile Devices	Activity Recognition	Carry Position	Gesture Recognition	Pedometer	Fall Detection
Activity Tracking for Wrist Devices	Activity Recognition for Wrist	Pose Estimation	Motion Intensity Detection	Standing vs Sitting Desk Detection	Fitness Activities
					Active Time
					Pedometer
					Sleep Monitoring

Cette étape est cruciale car elle nous permet d'installer les pilotes nécessaires pour le bon fonctionnement des composants MEMS présents sur notre carte d'évaluation, tels que l'accéléromètre, le gyroscope, le magnétomètre, le capteur de température, le capteur d'humidité, etc. Ces pilotes, fournis par ST Microelectronics, sont spécifiquement conçus pour interagir avec les composants MEMS et fournir une interface de programmation facile à utiliser pour les développeurs.

En installant le X-CUBE-MEMS1, nous intégrons les bibliothèques logicielles nécessaires dans notre environnement de développement, ce qui simplifie le

processus de programmation et nous permet d'accéder facilement aux fonctionnalités avancées des capteurs MEMS. Ces pilotes offrent une interface de haut niveau pour interagir avec les capteurs, ce qui permet aux développeurs de se concentrer sur la logique de leur application plutôt que sur les détails de bas niveau de la communication avec les capteurs.

The screenshot shows a software interface titled "STM32Cube MCU Packages and embedded software packs releases". At the top, there are tabs for "Software Packs" and "Pinout". Below the tabs, a message says "Releases Information was last refreshed 18 hours ago." There is a "+" button to add packages and a "-" button to remove them. A header row lists package providers: SEGGER, WES, emotas, portGmbH, and wolfSSL. Under "SEGGER", it says "STM32Cube MCU Packages". Under "STMicroelectronics", it says "STM32Cube MCU Packages". Under "Infineon", it says "ITTA_DB".

	Status	Description	Available
+	Green	X-CUBE-MEMS1	10.0.0
+	Green	Drivers and sample applications for MEMS components	10.0.0
+	Blue	Drivers and sample applications for MEMS components (S)	9.6.0

A "Details" section is expanded, showing a large empty area for displaying detailed information about the selected package.

At the bottom, there are buttons for "From Local ...", "From Url ...", "Refresh", "Install", "Remove", and "Close".

fig.16: Pilotes MEMS

Après cette étape, nous sélectionnons le type de composant, en l'occurrence l'IKS01A3, puis nous téléchargeons la bibliothèque associée.

Device MEMS1_Applications		10.0.0	
Board Part AccGyr		5.5.0	
Board Part AccMag		5.6.0	
Board Part Acc		1.4.0	
Board Part AccTemp / LIS2DTW12			Not selected
Board Part Mag		5.4.0	
Board Part HumTemp		5.6.0	
Board Part PressTemp		5.6.0	
Board Part Temp		1.4.0	
Board Part Gyr / A3G4250D			Not selected
Board Part PressTempQvar		1.2.0	
Board Part AccGyrQvar		1.4.0	
Board Part AccQvar / LIS2DUXS12			Not selected
Board Part AccGyrlspu / LSM6DSO16IS			Not selected
Board Part Gas / SGP40			Not selected
Board Extension IKS01A3	✓	1.12.0	✓
Board Extension IKS02A1		1.5.0	□
Board Extension IKS4A1		1.0.0	□
Board Support Custom		10.0.0	
Sensors STM32_MotionID_Library / Core		2.4.1	□
Sensors STM32_MotionFX_Library / Core		2.8.0	□
Sensors STM32_MotionOC_Library / Core		2.6.0	□

fig.17: sélection de l'extension iks01a3

Pour afficher les résultats sur un écran LCD, nous avons configuré le bus I2C2 avec les broches PB10 (SCL) et PB11 (SDA). Nous avons observé qu'il y avait un conflit lors de l'affichage sur l'écran LCD, car la communication des capteurs de l'IKS01A3 se faisait via le bus I2C1. Pour éviter ce conflit, nous avons décidé de travailler avec deux bus I2C distincts.

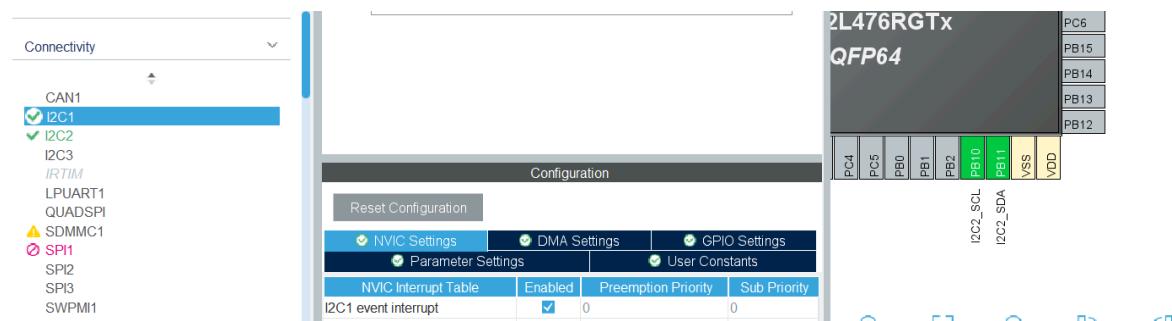


fig.18: Configuration I2C2

Grâce aux bibliothèques ajoutées, nous pouvons maintenant inclure deux bibliothèques fondamentales :

```
#include "iks01a3_env_sensors.h"  
  
#include "iks01a3_env_sensors_ex.h"
```

Dans la bibliothèque "iks01a3_env_sensors.h", nous avons accès à des fonctions nous permettant d'initialiser un capteur, de l'activer si l'initialisation s'est bien déroulée, et enfin de récupérer la valeur du capteur. Ensuite, nous avons affiché le résultat sur notre écran LCD. Pour notre cas, nous avons choisi d'utiliser le capteur STTS751, qui est un capteur de température.

```
if (IKS01A3_ENV_SENSOR_Init(IKS01A3_STTS751_0, ENV_TEMPERATURE) == HAL_OK)  
{  
    IKS01A3_ENV_SENSOR_Enable(IKS01A3_STTS751_0, ENV_TEMPERATURE);  
}  
  
/* Infinite loop */  
float temperature = 0.0f;  
while (1)  
{  
    /* Read the temperature value from the sensor */  
    IKS01A3_ENV_SENSOR_GetValue(IKS01A3_STTS751_0, ENV_TEMPERATURE, &temperature);  
    char tempString[20];  
    sprintf(tempString, "Temp: %.2f C", temperature);  
    clearlcd();  
    lcd_position(&hi2c2, 0, 0);  
    lcd_print(&hi2c2, tempString);  
    HAL_Delay(1000);
```

fig.19: main.c Avec la configuration du capteur de température STTS751

6.1 Câblage électrique:

Nous avons connecté notre microcontrôleur L476RG à l'IKS01A3, qui à son tour est connecté au Nucleo Shield L476RG, utilisé dans notre cas pour connecter l'écran LCD. Pour le LCD, nous avons utilisé le bus I2C2, ce qui nous a nécessité de dénuder les fils pour pouvoir les connecter aux bonnes broches : le PB10 (SCL) a été branché au D7, le PB11 (SDA) au D10, et le GND et VDD aux broches correspondantes. Toutes ces connexions ont été réalisées

en se référant à la documentation suivante:

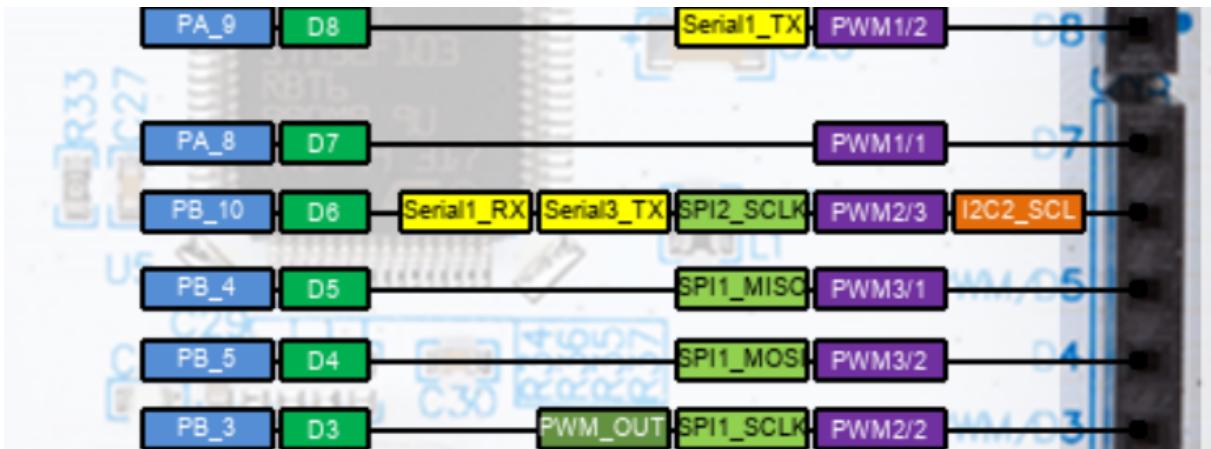


fig.20: Connection du I₂C2_SCL

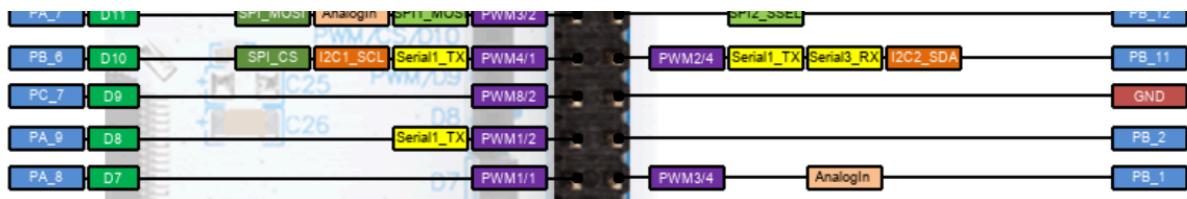
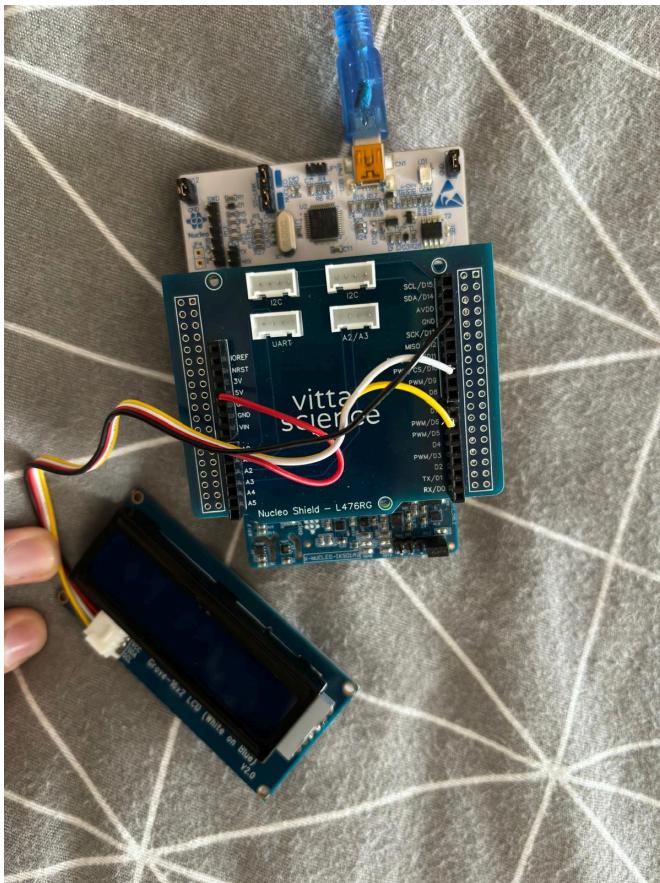


fig.21: Connection du I₂C2_SDA



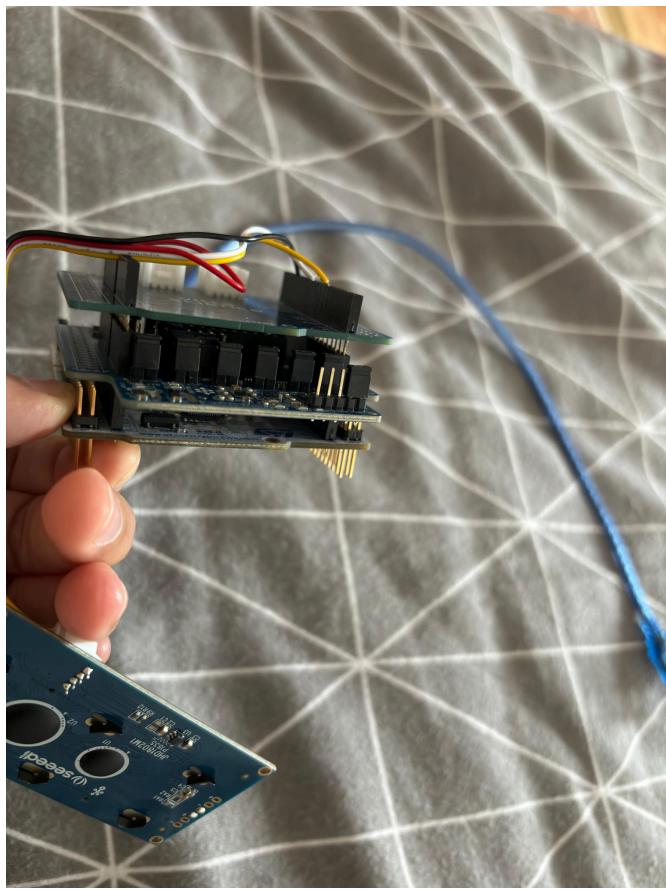




fig.22: Affichage de la température

7. Conclusion :

Le projet dans son ensemble a été une exploration passionnante 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, en configurant les cartes STM32, en connectant les capteurs et en établissant des communications entre les différents composants.

Ensuite, nous avons progressé vers des tâches plus complexes, telles que l'utilisation des modules LoRa pour établir une communication à longue portée entre les capteurs et les microcontrôleurs. Nous avons surmonté des défis techniques, comme la gestion des conflits de bus I2C et la sélection des bons paramètres de communication pour assurer une transmission fiable des données.

L'intégration du X-NUCLEO-IKS01A3 a enrichi notre projet en nous fournissant une gamme étendue de capteurs environnementaux, ce qui nous a permis de collecter des données précieuses sur la température, l'humidité, la pression et les mouvements.

Enfin, nous avons consolidé nos connaissances en développant des compétences pratiques en programmation, en utilisant des bibliothèques logicielles et en manipulant les données des capteurs pour les afficher sur des écrans LCD.

Dans l'ensemble, ce projet nous a offert une expérience complète dans la conception, le développement et le déploiement de systèmes embarqués pour des applications réelles, et nous a préparés à relever des défis similaires à l'avenir.

8. Annexes :

- Documentation LoRa: <https://RAK811 Lora Module Datasheet V1.4>
- Documentation ST Nucleo L476RG:
https://developer.nordicsemi.com/nRF Connect SDK/doc/1.4.99-dev1/zephyr/boards/arm/nucleo_l476rg/doc/index.htm
- Documentation IKS01A3:
<https://www.st.com/en/ecosystems/x-nucleo-iks01a3.html>