



---

**Partie 1:**  
**Compte Rendu de TP de base**  
**UE BE**

---

**Auteurs : Akel Fadwa - Khemchane ikram**

**Groupe : M1 SME Groupe AP**

# Sommaire

1. Introduction.....
1.1 Utilisation de STM32CubeIDE pour le développement d'applications embarquées.....
2. Utilisation du capteur DHT22.....
2.1 La carte nucleo STM32L476RG.....
2.2 Le schéma de câblage.....
2.3 Le câblage.....
2.4 Interprétation des données du capteur DHT22.....
2.5 Configuration et initialisation des pins.....
3. Résultat.....
4. Utilisation du capteur SHT31.....
5. Conclusion.....
6. Annexes.....

## 1. Introduction:

Lors de cette première partie des travaux pratiques, notre objectif était d'explorer et de comparer deux capteurs : le DTH22 et le STH31. Ces dispositifs, spécialisés dans la mesure de l'humidité et de la température, représentent des outils cruciaux dans divers domaines, allant de l'agriculture à la domotique, par exemple. Après avoir examiné les caractéristiques et les performances de ces capteurs, nous nous apprêtons désormais à entamer une nouvelle phase de notre apprentissage. Notre prochaine étape consistera à élaborer un projet (BE) mettant en œuvre un capteur LoRa, permettant ainsi une surveillance à distance et une transmission efficace des données. Ce projet marque une progression significative dans notre compréhension et notre application des technologies de capteurs, ouvrant la voie à des solutions innovantes pour diverses applications pratiques.

### 1.1 Utilisation de STM32Cube IDE pour le développement d'applications embarquées

STM32Cube IDE est un logiciel complet développé par STMicroelectronics. Il a été conçu pour simplifier la programmation et le développement d'applications destinées aux microcontrôleurs de la famille STM32. En tant qu'environnement de développement intégré (IDE), STM32Cube IDE offre une suite complète d'outils pour configurer les microcontrôleurs, générer du code et déboguer les applications. Tous ces outils sont regroupés dans une seule plateforme, ce qui simplifie considérablement le processus de création d'applications embarquées.

En plus de ces fonctionnalités, STM32Cube IDE intègre également un certain nombre de bibliothèques logicielles et de pilotes matériels qui facilitent l'interaction avec les périphériques intégrés aux microcontrôleurs STM32. Cela inclut des pilotes pour les interfaces de communication comme l'I2C, SPI, et UART, ainsi que pour les capteurs, les convertisseurs analogiques-numériques, et bien d'autres.

De plus, STM32Cube IDE prend en charge une variété de langages de programmation, y compris le C et le C++, ce qui permet aux développeurs de choisir le langage qui convient le mieux à leur application. Il offre également une interface utilisateur intuitive et facile à utiliser, avec des fonctionnalités de débogage avancées qui aident les développeurs à identifier et à résoudre rapidement les problèmes dans leur code.

## 2. Utilisation du Capteur DHT22 :

- Matériel utilisé :

Carte Nucleo STM32L476RG  
Un écran LCD JHD1802M1  
Un Shield Base  
Un Picoscope

### 2.1 La carte nucleo STM32L476RG :

Cette carte, au cœur de ce TP, possède une mémoire interne qui permet d'accueillir un programme informatique conçu par l'utilisateur. Ce programme consiste en une série d'instructions à exécuter par le microcontrôleur.

La carte est équipée d'une puce électronique, plus précisément un microcontrôleur STM32, qui est programmé à l'aide d'un ordinateur. Cette carte est particulièrement adaptée aux applications embarquées en raison de sa faible consommation d'énergie et de ses nombreuses fonctionnalités. Outre sa mémoire interne, la carte Nucleo STM32L476RG dispose également d'une variété de périphériques intégrés, comprenant des interfaces de communication et des convertisseurs analogiques-numériques. Ces

fonctionnalités offrent aux développeurs la possibilité de concevoir des applications complexes et puissantes.

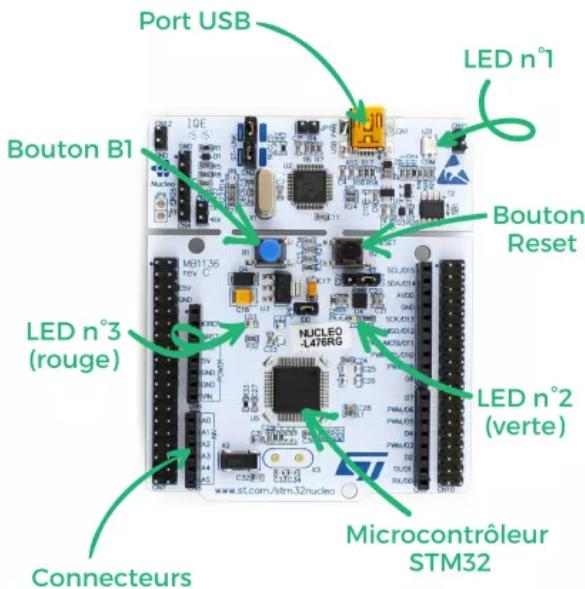


fig. 1 : la carte nucleo STM32L476RG

## 2.2 Le schéma de câblage :

À l'aide du logiciel Fritzing, nous avons élaboré notre schéma de câblage, visible dans la figure suivante :

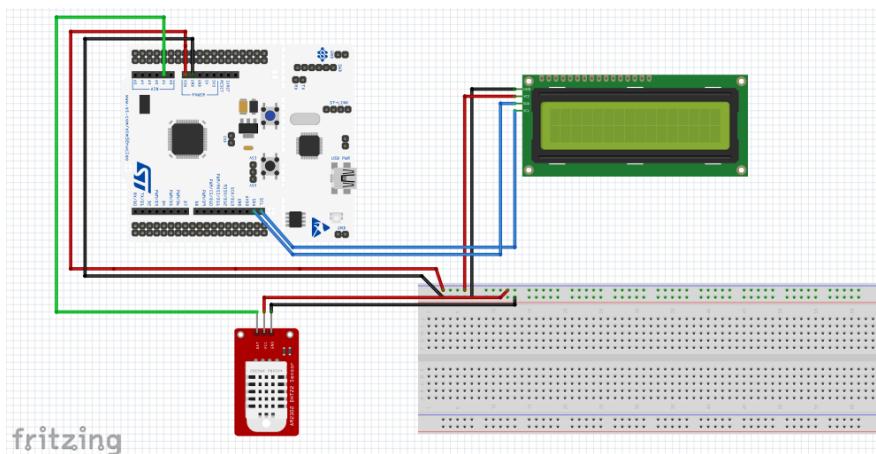


fig.2 : le schéma de câblage DHT22

### 2.3 Le câblage :

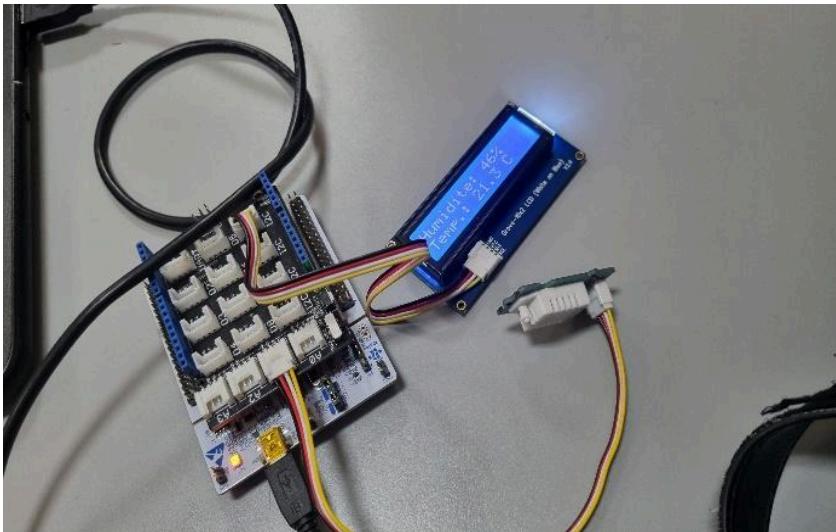


fig.3 Câblage finale

### 2.4 Interprétation des données du capteur DHT22:

Pour utiliser le capteur DHT22, qui mesure la température et l'humidité avec la carte STM32, nous avons dû nous référer à la documentation du fabricant pour comprendre son fonctionnement.

Le circuit fournit 16 bits pour l'humidité relative, exprimée en dixièmes et codée en binaire. Par exemple, si les 2 premiers bytes sont 0000 0010 1001 0010 = 0x0292, la conversion en décimal donne 658, ce qui correspond à une humidité de 65,8%.

Pour la température, les 2 bytes suivants sont 1000 0000 0110 0101. Le premier bit indique que la température est négative, et sa valeur absolue suit : 0000 0000 0110 0101 = 0x0065. La conversion en décimal donne 101, ce qui signifie que la température est de -10,1 degrés Celsius.

Le capteur envoie 5 mots de 8 bits, le cinquième mot étant la somme des quatre premiers.

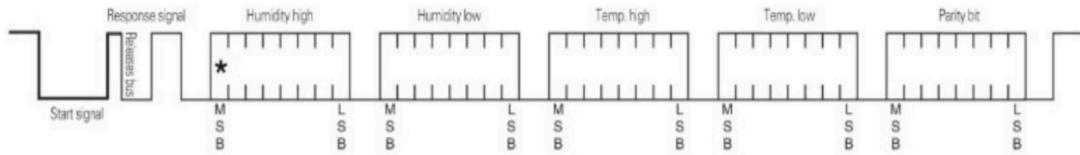
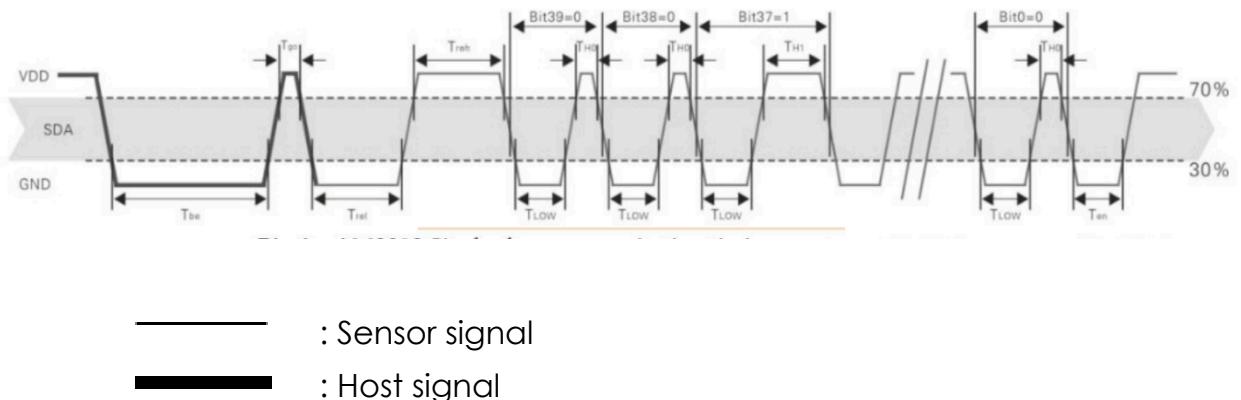


fig.4 : Chronologie de communication sur le bus unique

Chaque bit est transmis sous forme d'une impulsion positive de 50 µs pour un zéro, et de 26 µs pour un, suivie d'un intervalle négatif de 50 µs. Bien entendu, ces valeurs sont assorties de marges pour garantir la fiabilité de la transmission des données.



Symbol	Parameter	min	typ	max	Unit
$T_{be}$	Host the start signal down time	0.8	1	20	mS
$T_{go}$	Bus master has released time	20	30	200	µs
$T_{rel}$	Response to low time	75	80	85	µs
$T_{reh}$	In response to high time	75	80	85	µs
$T_{low}$	Signal "0", "1" low time	48	50	55	µs
$T_{H0}$	Signal "0" high time	22	26	30	µs
$T_{H1}$	Signal "1" high time	68	70	75	µs
$T_{en}$	Sensor to release the bus time	45	50	55	µs

fig.6 : Caractéristiques du signal à bus unique :

## 2.5 Configuration et initialisation des pins :

Lors de la phase de câblage, nous avons utilisé le logiciel STM32CubeMX pour configurer les broches nécessaires à notre projet et générer du code en langage C. Ensuite, nous avons perfectionné le programme dans l'IDE STM32Cube en y apportant les ajustements requis, en nous référant au guide Nucleo. L'intégralité du code est accessible sur notre dépôt GitHub.

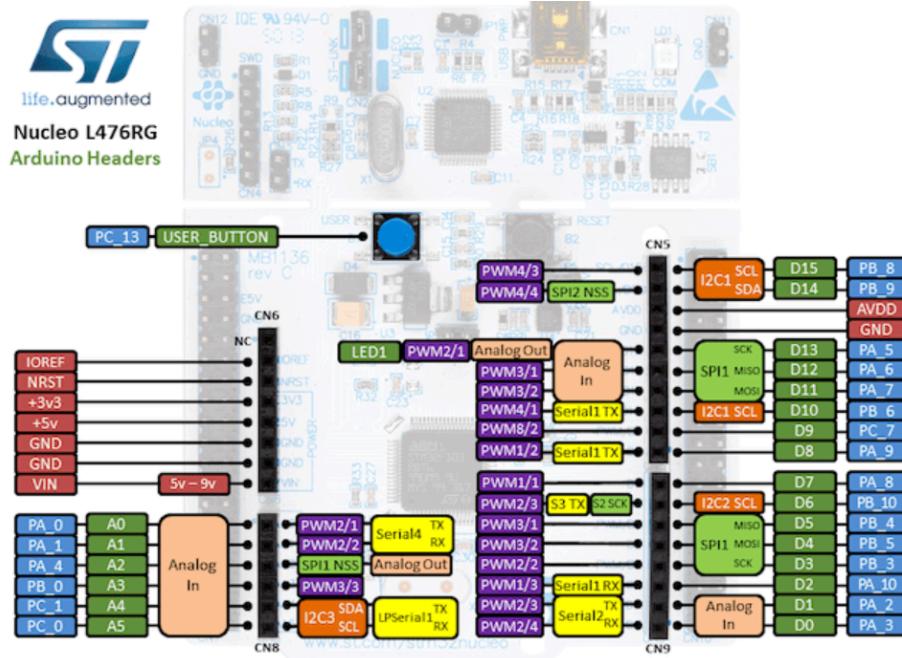


Fig.7 : Guide de la carte STM32

La configuration a été réalisée dans le logiciel, avec l'aide du guide, et a impliqué une série d'analyses pour générer du code. Nous avons connecté les broches PB8 et PB9 à la carte LCD pour configurer l'I2C, puis nous avons défini les broches PA1 en tant que GPIO\_ANALOG ou nous avons connecté le pin de données du DHT22, comme illustré dans la figure ci-dessous :

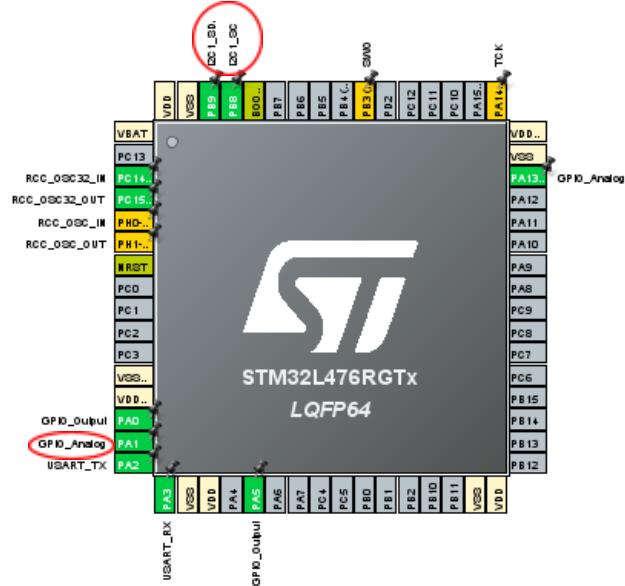


fig.8 : Configuration pins du STM32

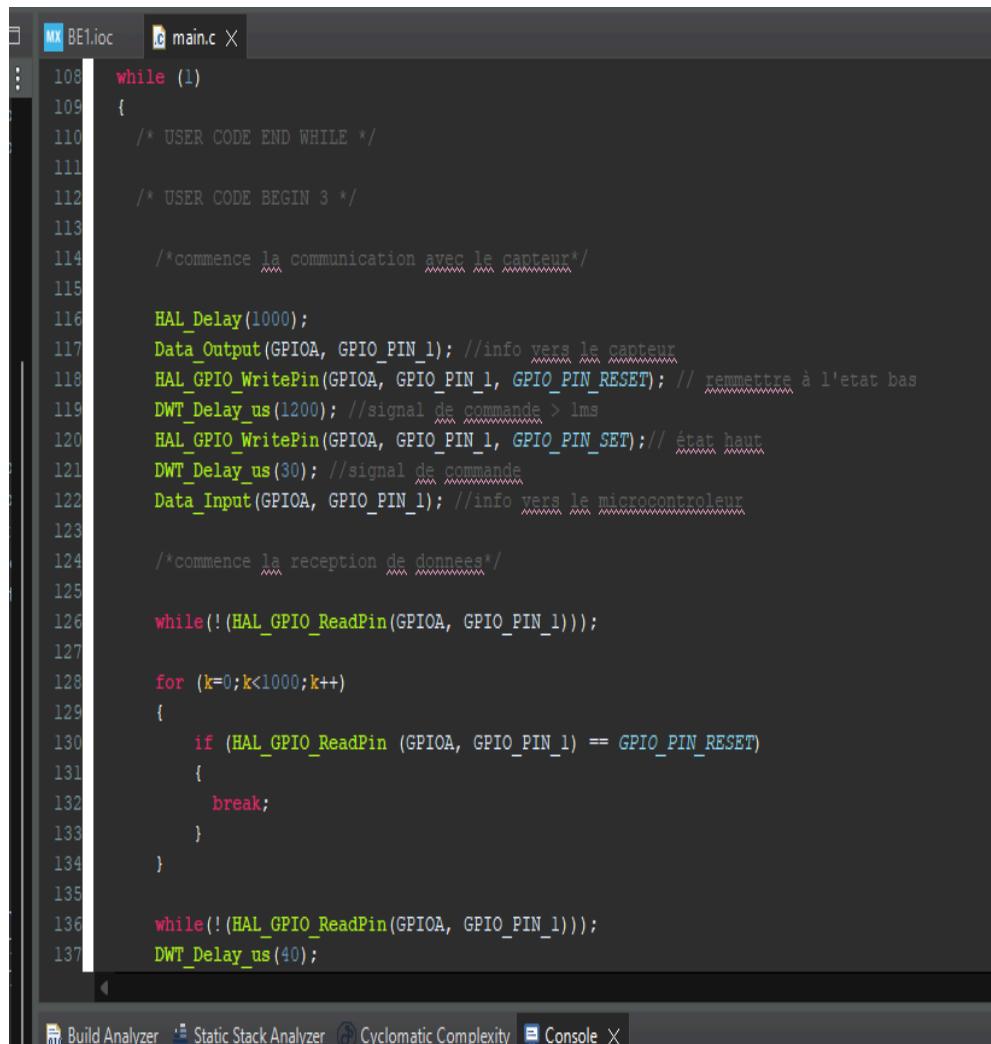
## 2.6 Visualisation des Résultats à travers le code :

## 1. Le Code :

Vous trouverez ci-dessous un extrait de code, tandis que la version complète est disponible sur notre compte GitHub.

```
/* USER CODE BEGIN EFP */
...
/* USER CODE END EFP */
...
/* Private defines */
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
...
/* USER CODE BEGIN Private defines */
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_5
#define USART_RX_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
...
/* USER CODE END Private defines */

```



```

MX BE1.ioc main.c X

108 while (1)
109 {
110     /* USER CODE END WHILE */
111
112     /* USER CODE BEGIN 3 */
113
114     /*commence la communication avec le capteur*/
115
116     HAL_Delay(1000);
117     Data_Output(GPIOA, GPIO_PIN_1); //info vers le capteur
118     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // remettre à l'état bas
119     DWT_Delay_us(1200); //signal de commande > lms
120     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET); // état haut
121     DWT_Delay_us(30); //signal de commande
122     Data_Input(GPIOA, GPIO_PIN_1); //info vers le microcontrôleur
123
124     /*commence la réception de données*/
125
126     while(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1));
127
128     for (k=0;k<1000;k++)
129     {
130         if (HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_1) == GPIO_PIN_RESET)
131         {
132             break;
133         }
134     }
135
136     while(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1));
137     DWT_Delay_us(40);

```

Build Analyzer Static Stack Analyzer Cyclomatic Complexity Console X

fig.8 : Extrait du code en C .

## 2. La visualisation :

En utilisant un PicoScope, nous avons observé la trame de réponse du capteur DHT22, qui représente la réponse de notre capteur. Après l'avoir convertie d'abord en binaire puis en hexadécimal, nous obtenons les valeurs de température et d'humidité, conformément à la manière dont le capteur fonctionne. Vous pouvez visualiser cette procédure dans la figure suivante :

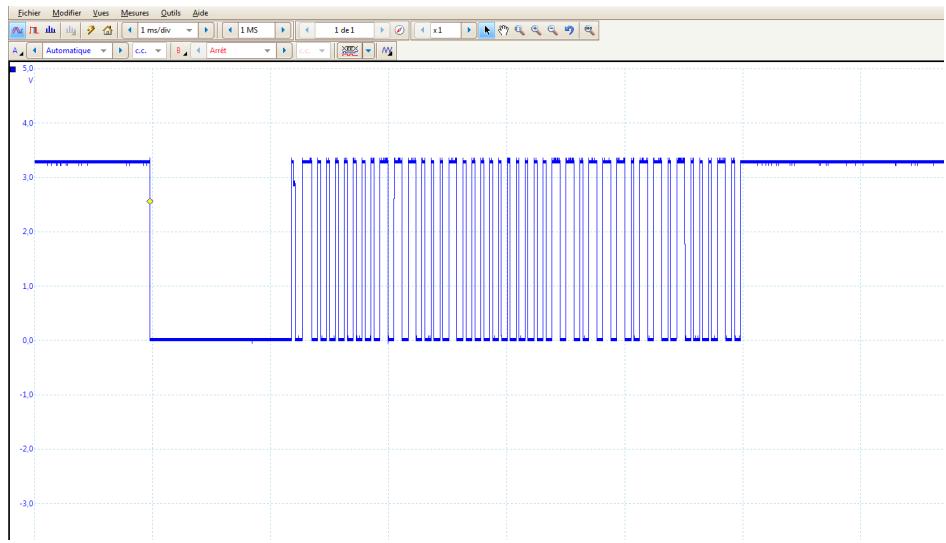


fig.10 : La trame correspondante au capteur DHT22

### 3. Résultat :

Comme le montre la figure, l'affichage des valeurs de température et d'humidité correspondantes à la pièce est clairement visible.

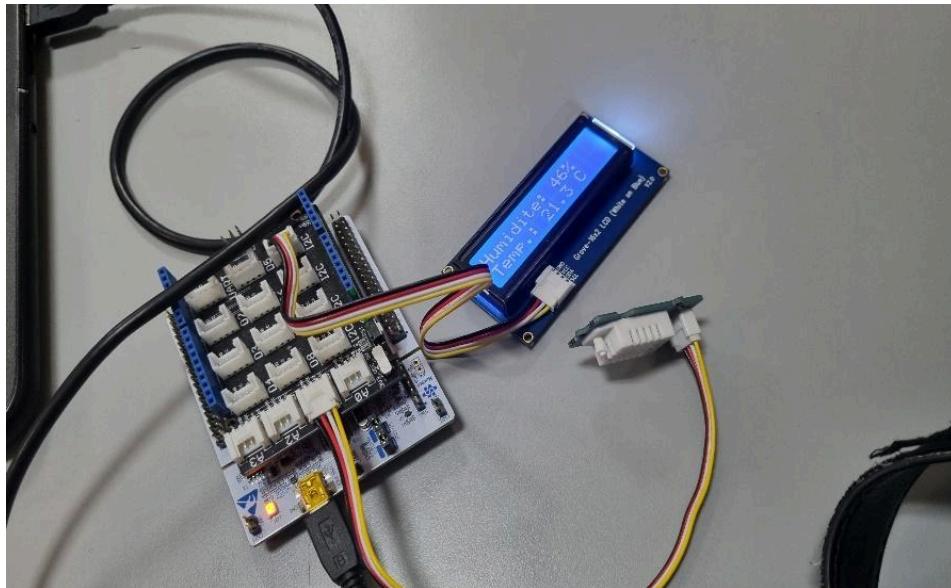


fig. 11: Affichage des Valeurs de Température et d'Humidité

#### 4. Utilisation du capteur SHT31:

Le capteur de température et d'humidité Grove - Temp&Humi (SHT31) est réputé pour sa fiabilité, sa précision et sa réponse rapide. Utilisant la technologie CMOSens® de Sensirion, la puce du capteur intégré est soigneusement calibrée, linéarisée et compensée pour assurer une sortie numérique précise.

Ce module affiche une précision typique de  $\pm 2\%$ RH pour l'humidité relative et de  $\pm 0.3^{\circ}\text{C}$  pour les mesures de température. Compatible avec des tensions de 3.3 Volts et 5 Volts, il élimine le besoin d'un décaleur de niveau de tension. La communication s'effectue via le bus série I<sup>2</sup>C, prenant en charge des vitesses allant jusqu'à 1 MHz. De plus, une bibliothèque hautement abstraite a été fournie pour simplifier son utilisation.

Ce capteur représente un outil précieux pour les applications nécessitant une surveillance précise de la température et de l'humidité. Ses performances robustes, associées à sa facilité d'utilisation et à sa compatibilité avec diverses plateformes, en font un choix idéal pour une large gamme de projets, de la surveillance environnementale à l'automatisation industrielle.

#### Specifications #

Parameter	Value
Input voltage (VCC)	3.3 volts or 5 volts
I/O Logic Level	3.3 volts or 5 volts based on VCC
Operating Current	100 $\mu\text{A}$
Operating Temperature	-40–125 °C
Temperature Sensor Range	-40–125 °C, with $\pm 0.3^{\circ}\text{C}$ accuracy
Humidity Sensor Range	0% - 100%(Relative Humidity), with $\pm 2\%$ accuracy
Sensor Chip	SHT31( <a href="#">Datasheet</a> )
Port	I <sup>2</sup> C
Weight	4 g (for breakout board), 9 g for whole package each piece
Dimensions	40(length)×20(width) mm

fig.12: Specifications

Pour notre projet, nous avons choisi d'utiliser un capteur Grove, qui se connecte directement sur le Base Shield Grove. Le Base Shield Grove est une carte d'extension conçue pour faciliter l'intégration de différents modules Grove avec les microcontrôleurs compatibles, tels que notre carte STM32.

Le Base Shield Grove fournit une variété de connecteurs, y compris des ports I2C, ce qui permet une connexion simple et rapide des capteurs compatibles Grove. De plus, le Base Shield Grove fournit une alimentation stable aux modules connectés, éliminant ainsi le besoin d'une alimentation externe pour chaque capteur. Cela rend notre système plus compact et plus facile à gérer.

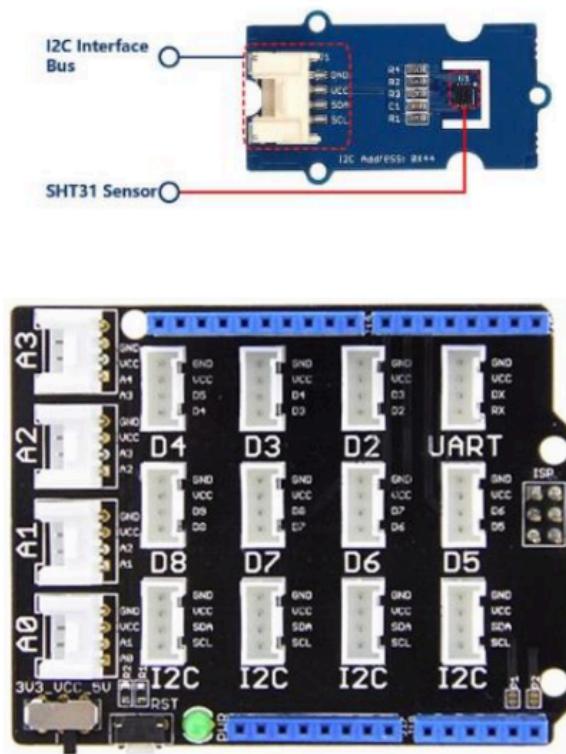
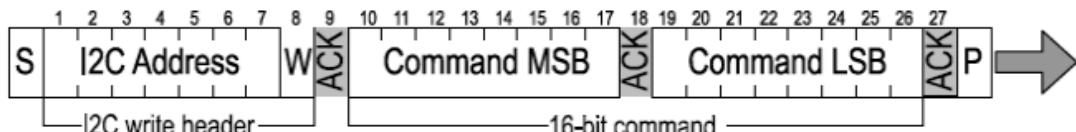


fig.13: Capteur SHT31 et la carte Base Shield Grove

Une fois que le capteur est correctement connecté au shield et alimenté, il est prêt à recevoir des commandes. La transmission débute par un signal de départ (START) conformément au protocole standard I2C. Ensuite, l'en-tête d'écriture en I2C est envoyé, comprenant 7 bits pour l'adresse (0x44 pour le SHT31) et un bit de lecture/écriture (0 pour l'écriture). Ce signal est suivi de 16

bits de commande de mesure, qui déterminent le type de mesure à effectuer.

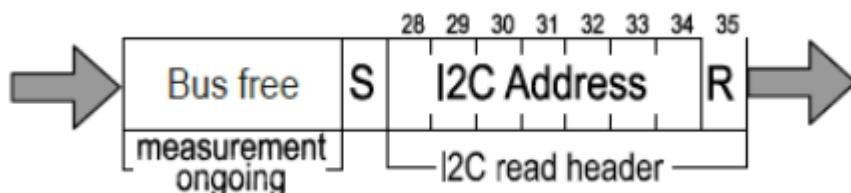
Il est important de noter que le SHT31 est équipé d'une fonction de mesure en continu, ce qui permet de simplifier la communication avec le capteur. Une fois que la commande de mesure est envoyée, le capteur effectue automatiquement la mesure et transmet les données correspondantes à la carte STM32 via le bus I2C.



Repeatability	Clock stretching	Hex. code	
		MSB	LSB
High	enabled	0x2C	06
Medium			0D
Low			10
High	disabled	0x24	00
Medium			0B
Low			16

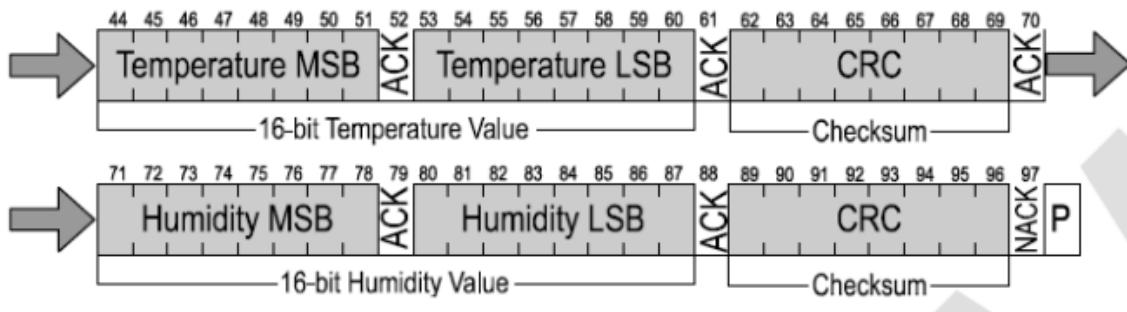
Dans notre configuration, nous avons opté pour l'utilisation de la commande 0x2416, désactivant ainsi la répétition faible et l'extension de l'horloge. Avant de commencer la mesure, le capteur envoie un bit ACK (bit d'acquittement) pour indiquer la réception des commandes. Une fois cet acquittement envoyé, le capteur démarre la mesure.

Pour lire les valeurs mesurées, nous envoyons une commande de lecture comprenant les 7 bits d'adresse du capteur, suivis d'un bit pour indiquer la lecture des données. Cette approche permet une communication efficace avec le capteur et garantit une récupération précise des données mesurées.



Après avoir envoyé la commande de lecture, le capteur transmet les données de mesure, comprenant 16 bits pour la température et 16 bits pour l'humidité. À la fin de chaque ensemble de données, un code de vérification CRC (Cyclic Redundancy Check) est inclus pour garantir l'intégrité des données transmises.

Le CRC est un mécanisme de détection d'erreurs largement utilisé qui permet de vérifier si les données ont été altérées pendant la transmission. Grâce à cette vérification, nous pouvons avoir confiance dans l'exactitude des mesures fournies par le capteur, même dans des environnements sujets à des interférences électromagnétiques ou à d'autres perturbations du signal.



Pour convertir les valeurs reçues en degrés Celsius et en humidité relative (RH), nous appliquons les formules suivantes :

**Relative humidity conversion formula (result in %RH):**

$$RH = 100 \cdot \frac{S_{RH}}{2^{16} - 1}$$

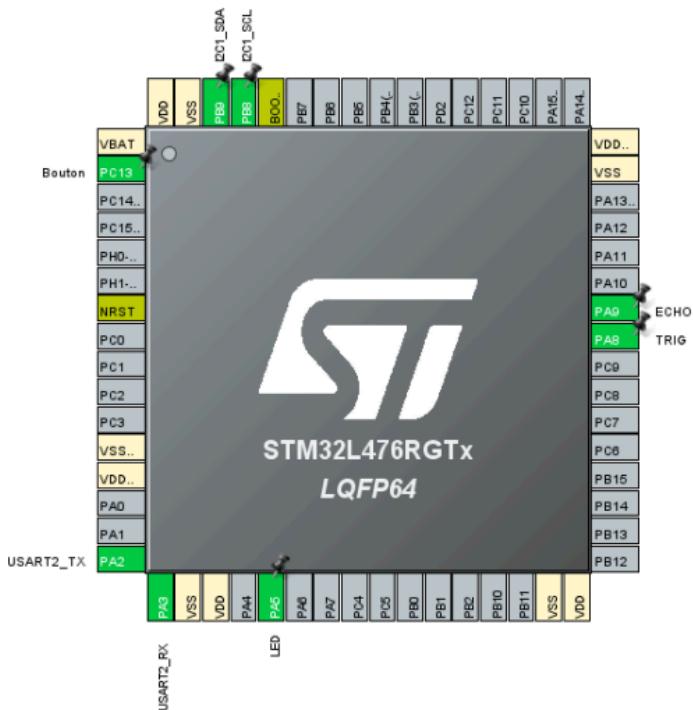
**Temperature conversion formula (result in °C & °F):**

$$T [{}^{\circ}\text{C}] = -45 + 175 \cdot \frac{S_T}{2^{16} - 1}$$

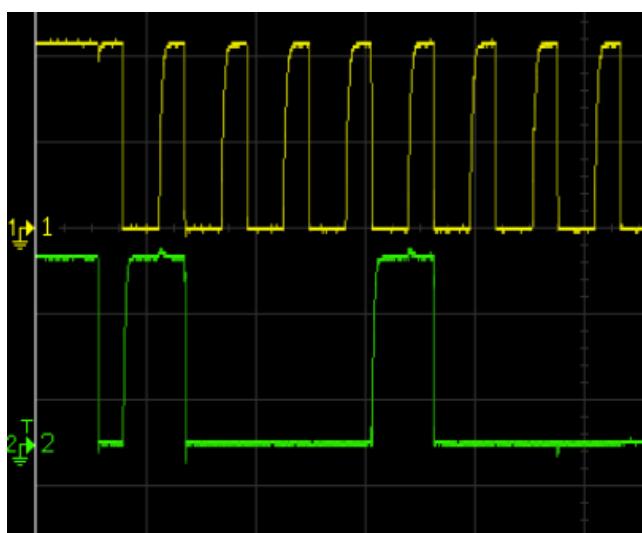
$$T [{}^{\circ}\text{F}] = -49 + 315 \cdot \frac{S_T}{2^{16} - 1}$$

Pour utiliser ce capteur, nous activons les broches PB9 et PB8 comme SDA et SCL respectivement dans CubeMX afin de communiquer avec les ports I2C disponibles sur le shield Grove. De plus, nous utilisons le bus UART pour communiquer avec un ordinateur et afficher les résultats.

Une fois les broches déclarées dans CubeMX, nous générerons le code et commençons à écrire les étapes nécessaires pour utiliser pleinement ce capteur. Cela comprend l'initialisation du capteur, la configuration des paramètres de mesure et la lecture des données mesurées.

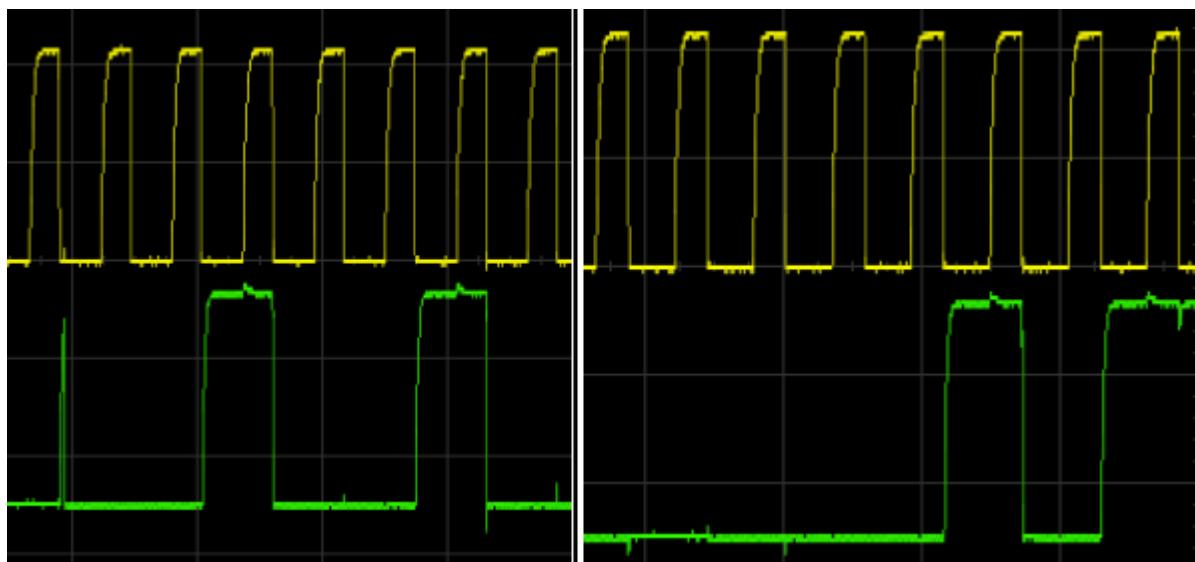


## Visualisation sur l'oscilloscope:

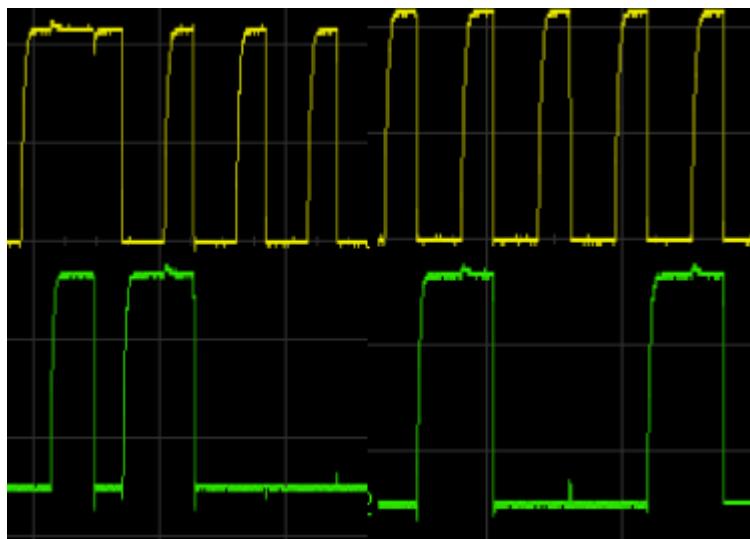


En utilisant l'oscilloscope, nous pouvons récupérer l'adresse du capteur, qui est affichée sous la forme 01000100, équivalente à 0x44 en hexadécimal (avec un décalage d'un bit sur l'oscilloscope). Cette adresse est cruciale pour établir la communication avec le capteur et lui envoyer des commandes spécifiques pour l'initialisation et la récupération des données mesurées. Grâce à cette méthode de surveillance, nous pouvons vérifier que le capteur répond correctement et que la communication s'établit comme prévu.

Suite à cette étape, nous récupérons les bits de poids fort (MSB) et de poids faible (LSB) de la commande 0X2416. Il est important de noter qu'un décalage d'un bit est observé sur l'oscilloscope lors de cette opération. En isolant ces bits, nous pouvons garantir que la commande est correctement interprétée par le capteur, ce qui est essentiel pour assurer des mesures précises de la température et de l'humidité.



Dans cette séquence, nous retrouvons d'abord le bit de départ (start bit), suivi de l'adresse du capteur, et enfin le bit de commande de lecture. Le start bit indique le début de la transmission de données, l'adresse identifie spécifiquement le capteur SHT31, et le bit de commande de lecture permet d'indiquer au capteur que nous souhaitons lire les données actuellement mesurées.



Suite au calcul de la température, nous obtenons le résultat suivant : 0110 0111 0101 1110, équivalent à 26462 en décimal. En utilisant cette valeur, nous pouvons déterminer la température de la pièce au moment de la mesure en utilisant la formule suivante :

$$T = -45 + 175 * (26462 / (2^{16} - 1))$$

Après avoir effectué le calcul, nous trouvons que la température de la salle au moment de la mesure est de 25.66°C. Ce résultat représente une estimation précise de la température actuelle, ce qui est essentiel pour une surveillance précise de l'environnement.



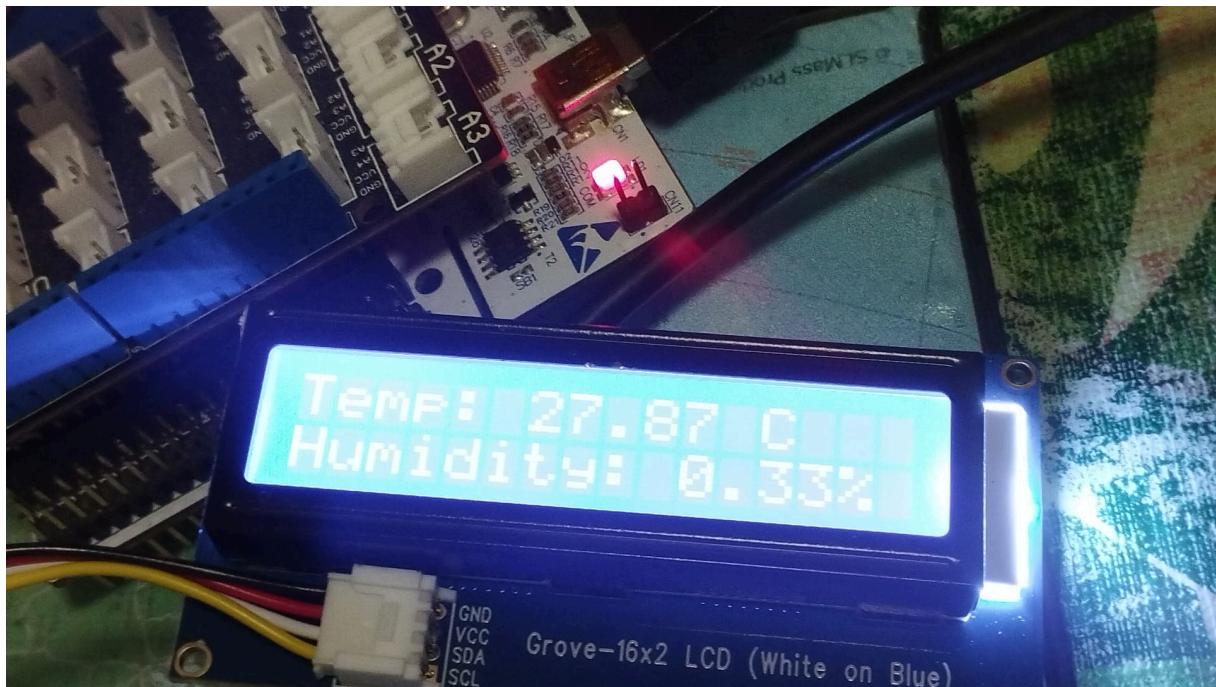


fig.14: Affichage du résultat final

## 5. Conclusion:

Dans cette première phase de notre projet, nous avons réalisé la configuration des capteurs SHT31 et DHT22, ainsi que l'affichage des données sur un écran LCD à l'aide de deux cartes STM32. Cette étape nous a permis de renforcer nos compétences dans l'utilisation des cartes STM32 et de comprendre le fonctionnement de différents composants et protocoles de communication. Notre objectif principal était de comparer les résultats obtenus avec les informations fournies par le fabricant dans les datasheets. Cette expérience nous a non seulement permis de confirmer les fonctionnalités des capteurs, mais aussi d'approfondir notre compréhension des spécifications techniques et des protocoles de communication associés à ces dispositifs. Ainsi, nous sommes mieux préparés pour la suite de notre projet, où nous explorerons davantage les capacités de ces capteurs et les intégrerons dans des applications pratiques.

**6. Annexes :**

- ST NUCLEO L476RG:  
[https://developer.nordicsemi.com/nRF\\_Connect\\_SDK/doc/1.4.99-dev1/zephyr/boards/arm/nucleo\\_l476rg/doc/index.html](https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.4.99-dev1/zephyr/boards/arm/nucleo_l476rg/doc/index.html)
- Usage Capteur DHT22:  
<https://controllerstech.com/temperature-measurement-using-dht22-in-stm32/>
- <https://www.didel.com/DHT22.pdf>
- Grove - 16x2 LCD: [https://wiki.seeedstudio.com/Grove-16x2\\_LCD\\_Series/](https://wiki.seeedstudio.com/Grove-16x2_LCD_Series/)
- Capteur SHT31:  
[https://wiki.seeedstudio.com/Grove-TempAndHumi\\_Sensor-SHT31/](https://wiki.seeedstudio.com/Grove-TempAndHumi_Sensor-SHT31/)
-