UNIVERSITÉ TOULOUSE III PAUL SABATIER

Université Fédérale Toulouse Midi-Pyrénées

FACULTÉ SCIENCES ET INGÉNIERIE

UE KEAS8AF1 - Réalisations systèmes et microsystèmes

RAPPORT DE TRAVAUX PRATIQUE

RÉALISATION DES SYSTÈMES BE

Élèves :
Rayan **SARDOU**
Amayas **KACI**

Enseignant :
Thierry **PERISSE**
thierry.perisse@univ-tlse3.fr

Année universitaire 2024/2025

Table des matières

Table des figures

Listings

I Introduction

Contexte général

Ce rapport présente la mise en oeuvre de capteurs environnementaux dans le cadre d'un projet de **ruche connectée**. L'objectif principal est de mesurer en temps réel des données de température, humidité et poids, puis de les transmettre à distance via un module **LoRa-E5**. La plateforme matérielle repose sur un microcontrôleur **STM32**, utilisant les protocoles **I2C**, **UART** et une interface **GPIO** pour dialoguer avec les capteurs **SHT31**, **DHT22**, **HX711** et le module LoRa.

1 Capteur SHT31

1.1 Analyse de la documentation constructeur

Le capteur de température et d'humidité SHT31 est documenté en détail dans la fiche technique accessible via le lien suivant : Datasheet_SHT3x_DIS.pdf.

Transmission de données L'échange de données avec le capteur suit un protocole bien défini basé sur la communication I2C. L'adresse que nous devons transmettre est encodée sur 3 octets, structurés comme suit :

- 7 bits pour l'adresse I2C du capteur
- 1 bit indiquant l'opération (lecture ou écriture)
- 8 bits correspondant au MSB de la commande
- 8 bits correspondant au LSB de la commande

Condition		Hex. code	
Repeatability	mps	MSB	LSB
High	0.5	0x20	32
Medium			24
Low			2F
High	1	0x21	30
Medium			26
Low			2D
High	2	0x22	36
Medium			20
Low			2B
High	4	0x23	34
Medium			22
Low			29
High	10	0x27	37
Medium			21
Low			2A
e.g. 0x2130: 1 high repeatability mps - measurement per second			

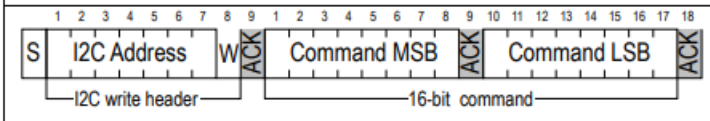

FIGURE I.1 – Structure des données transmises

Le choix de la valeur MSB permet de configurer la fréquence des mesures, c'est-à-dire le nombre de mesures effectuées par seconde. De son côté, la valeur LSB détermine la précision de la mesure, appelée aussi répétabilité.

Dans notre projet, nous avons adopté les paramètres suivants :

— Fréquence d'acquisition : 1 mesure par seconde (MSB = 0x21)

— Précision : Medium (LSB = 0x26)

Concernant l'adresse I2C du capteur, nous utilisons l'adresse par défaut définie dans la documentation du fabricant : 0x44.

SHT3x-DIS	I2C Address in Hex. representation	Condition
I2C address A	0x44 (default)	ADDR (pin 2) connected to logic low
I2C address B	0x45	ADDR (pin 2) connected to logic high

FIGURE I.2 – Adresses I2C disponibles pour le capteur SHT31

Réception de données Lorsqu'une requête est envoyée au capteur, celui-ci répond en transmettant une trame de données contenant les informations mesurées. La structure de la réponse est la suivante :

— 7 bits pour l'adresse I2C du capteur
— 1 bit indiquant qu'il s'agit d'une lecture
— 16 bits pour la valeur de température (8 bits MSB suivis de 8 bits LSB)
— 8 bits pour le code de vérification CRC (Checksum)
— 16 bits pour la valeur d'humidité (8 bits MSB suivis de 8 bits LSB)
— 8 bits pour le CRC de l'humidité

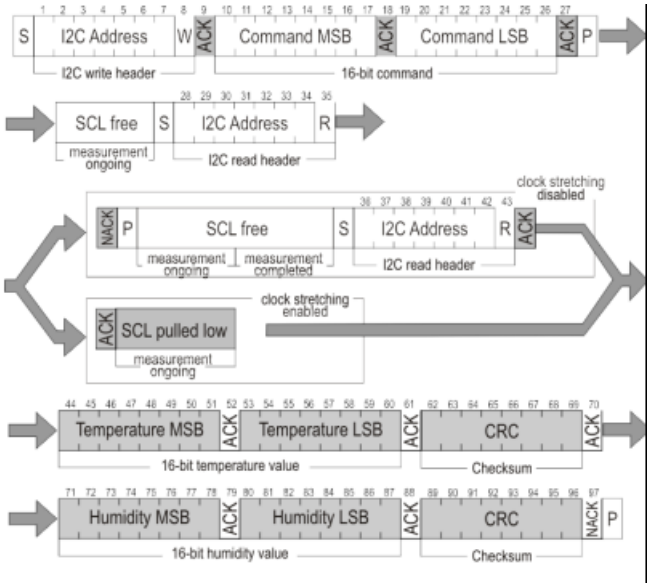

FIGURE I.3 – Structure de la trame de réponse envoyée par le capteur

Chaque mesure (température et humidité) est donc accompagnée d'un code de contrôle d'intégrité (CRC) sur 8 bits, permettant de vérifier que les données reçues ne contiennent pas d'erreur de transmission.

Analyse des trames I2C avec le Picoscope Afin de vérifier le bon fonctionnement du capteur SHT31 et de s'assurer que le protocole I2C est bien respecté, nous avons

capturé et analysé les trames échangées entre le microcontrôleur STM32 et le capteur à l'aide d'un oscilloscope Picoscope.

Émission des commandes vers le capteur L'envoi de commandes I2C au capteur est réalisé dans la boucle `while` du fichier `main.c` en utilisant la fonction suivante :

```
while(1){
    HAL_I2C_Master_Transmit(&hi2c1, (0x44 << 1), commande, 2, 5000);
    HAL_Delay(100);
}
```

Listing 1 – Envoi de commandes I2C au capteur

Explication des paramètres :
- `0x44 « 1` : Adresse I2C du capteur avec le bit d'écriture.
- `commande` : Tableau contenant les octets de commande à envoyer (MSB et LSB).
- `2` : Nombre d'octets envoyés.
- `5000` : Timeout en millisecondes.
- Cette commande configure la fréquence d'échantillonnage et la précision des mesures effectuées par le capteur.

Connexion du Picoscope pour l'analyse des trames Pour observer les signaux I2C échangés entre le STM32 et le capteur, nous avons connecté les sondes du Picoscope comme suit :
- Canal A (bleu) : connecté à la ligne SCL (horloge).
- Canal B (rouge) : connecté à la ligne SDA (données).
- GND (noir) : connecté à la masse de la carte STM32.

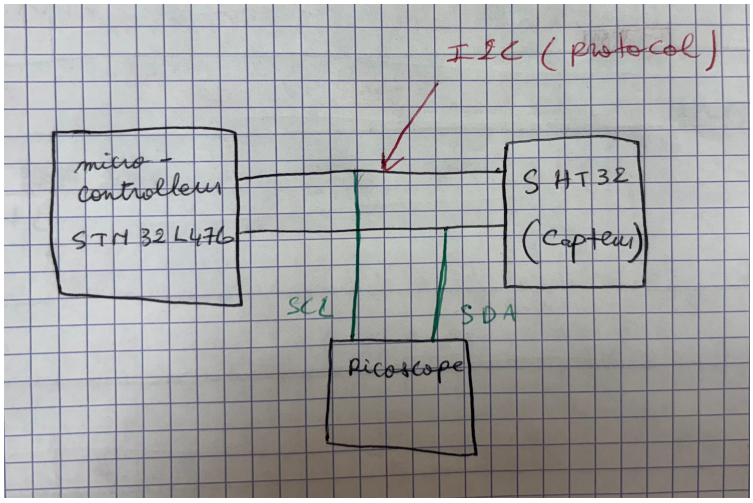

FIGURE I.4 – Connexion des sondes du Picoscope aux lignes I2C du STM32

Analyse de la trame de transmission L'image ci-dessous montre la trame capturée par le Picoscope lors de l'envoi de la commande au capteur :

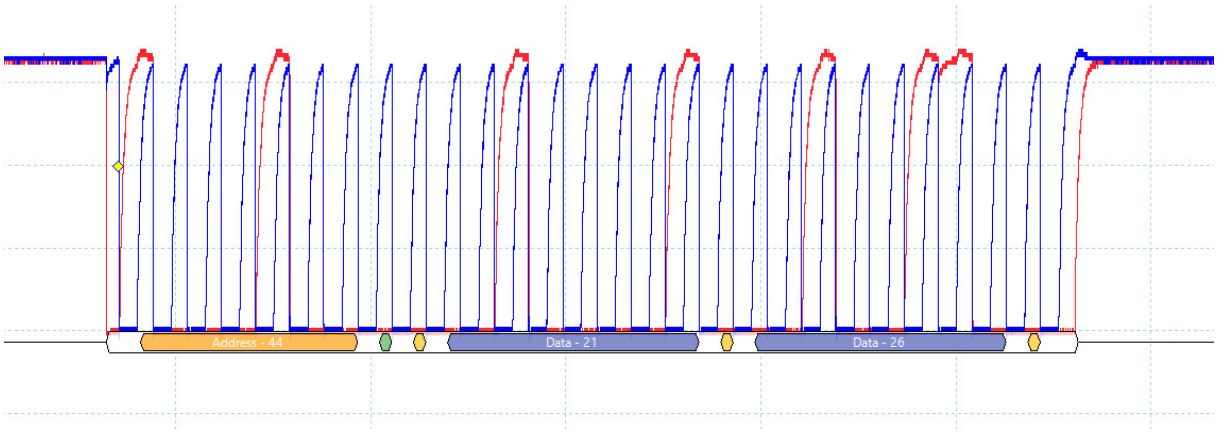

FIGURE I.5 – Trame I2C capturée lors de l'envoi de la commande au capteur

— Courbe bleue : signal d'horloge SCL.
— Courbe rouge : signal de données SDA.

Le logiciel du Picoscope décode automatiquement la trame I2C et affiche les valeurs en hexadécimal, facilitant l'analyse. Dans la capture ci-dessus, on observe :
— Adresse I2C : 0x44, confirmant la communication avec le capteur.
— Octets de commande envoyés : 0x21 et 0x26, correspondant à notre configuration (1 mesure/seconde et précision Medium).

Réception des données depuis le capteur Après l'envoi de la commande, nous effectuons une lecture des données mesurées par le capteur avec la fonction suivante :

```
while(1){
    HAL_I2C_Master_Receive(&hi2c1, (0x44 << 1), data, 6, 5000);
    HAL_Delay(100);
}
```

Listing 2 – Lecture des données I2C du capteur

Explication des paramètres :
— `0x44 « 1` : Adresse I2C du capteur avec le bit de lecture.
— `data` : Tableau contenant les octets de données reçus.
— `6` : Nombre d'octets à lire (température + humidité + CRC).
— `5000` : Timeout en millisecondes.

Analyse de la trame de réception L'image ci-dessous montre la trame capturée lors de la réception des données envoyées par le capteur :

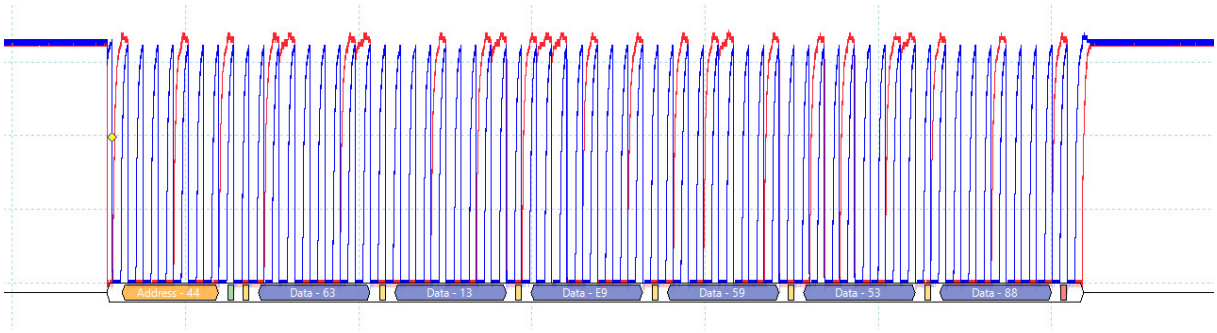

FIGURE I.6 – Trame I2C capturée lors de la lecture des mesures du capteur

Analyse de la trame reçue :
— Premier octet : Adresse I2C du capteur (0x44) avec le bit de lecture.
— Six octets suivants : Données transmises par le capteur :
  — 0x63 0x13 : Température mesurée.
  — 0xE9 : CRC de la température.
  — 0x59 0x53 : Humidité mesurée.
  — 0x88 : CRC de l'humidité.

Ces valeurs sont conformes aux spécifications du capteur et permettent de reconstituer la température et l'humidité après conversion.

2 Capteur DHT22

2.1 Analyse de la documentation constructeur

Le capteur de température et d'humidité DHT22 est documenté en détail dans la fiche technique accessible via le lien suivant : DHT22 Datasheet. Il est le successeur du DHT11 et offre une meilleure précision et une plage de mesure plus large.

Caractéristiques Techniques
- **Plage de Température** : Mesure de -40°C à 80°C avec une précision de ±0.5°C.
- **Plage d'Humidité** : Mesure de 0 à 100% avec une précision de ±2% à 5%.
- **Tension d'Alimentation** : Fonctionne avec une tension de 3 à 5 volts.
- **Sortie de Données** : Utilise une interface numérique à un seul fil pour transmettre les données.

Principe de Fonctionnement Le DHT22 utilise un capteur capacitif pour mesurer l'humidité et un thermistor NTC pour mesurer la température. Ces signaux analogiques sont convertis en signaux numériques avant d'être transmis via le pin de données.

2.2 Connexion et Utilisation

Pour connecter le DHT22, relier le premier pin à une alimentation de 3-5V, le deuxième pin au pin de données du microcontrôleur, et le troisième pin à la masse. Un résistor de pull-up de 4.7K à 10K ohms est souvent utilisé entre le pin de données et l'alimentation.

```
102
103   /* Infinite loop */
104   /* USER CODE BEGIN WHILE */
105   while (1)
106   {
107       HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_1);
108       delay_us(10);
109   }
110   /* USER CODE END 3 */
111 }
112
```

FIGURE I.7 – Exemple de code pour l'utilisation du DHT22 avec un microcontrôleur.

2.3 Protocole de Communication

Le DHT22 utilise un protocole de communication série simple :

1. **Initialisation** : Le microcontrôleur envoie un signal de départ en mettant le pin de données à bas pendant au moins 18 ms, puis à haut pendant 20-40 µs. 2. **Réponse du Capteur** : Le capteur répond avec un signal bas de 80 µs suivi d'un signal haut de 80 µs. 3. **Transmission des Données** : Le capteur envoie un train de 40 bits, incluant les données d'humidité et de température, suivi d'un bit de parité.

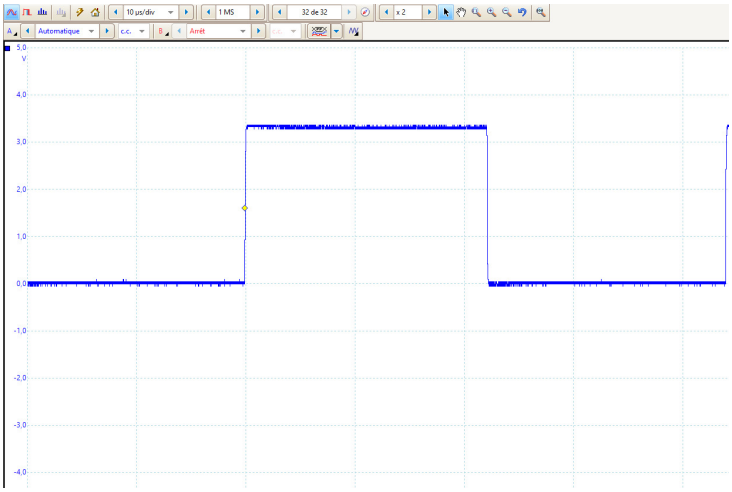

FIGURE I.8 – Signal capturé par un oscilloscope montrant la transmission des données.

2.4 Analyse des Trames

Chaque bit de données est représenté par un signal haut de 50 µs pour un "1" et de 26-28 µs pour un "0". Le microcontrôleur doit lire ces signaux à des intervalles précis pour décoder correctement les données.

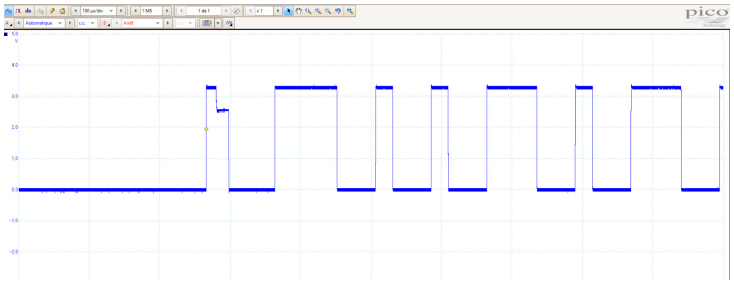

FIGURE I.9 – Trame de données capturée par un oscilloscope.

2.5 Exemple de Code

Voici un extrait de code pour l'interfaçage du DHT22 avec un microcontrôleur STM32 :

```
#define DHT22_PORT GPIOA
#define DHT22_PIN GPIO_PIN_0

void DHT22_Start(void) {
    Set_Pin_Output(DHT22_PORT, DHT22_PIN);
    HAL_GPIO_WritePin(DHT22_PORT, DHT22_PIN, 0);
    delay(18000);
    HAL_GPIO_WritePin(DHT22_PORT, DHT22_PIN, 1);
    delay(20);
    Set_Pin_Input(DHT22_PORT, DHT22_PIN);
}

uint8_t DHT22_Check_Response(void) {
    uint8_t Response = 0;
    delay(40);
    if (!(HAL_GPIO_ReadPin(DHT22_PORT, DHT22_PIN))) {
        delay(80);
        if ((HAL_GPIO_ReadPin(DHT22_PORT, DHT22_PIN)))
            Response = 1;
    }
    while ((HAL_GPIO_ReadPin(DHT22_PORT, DHT22_PIN)));
    return Response;
}

uint8_t DHT22_Read(void) {
    uint8_t i = 0, j;
    for (j = 0; j < 8; j++) {
        while (!(HAL_GPIO_ReadPin(DHT22_PORT, DHT22_PIN)));
        delay(40);
        if (!(HAL_GPIO_ReadPin(DHT22_PORT, DHT22_PIN)))
            i &= ~(1 << (7 - j));
        else
            i |= (1 << (7 - j));
        while ((HAL_GPIO_ReadPin(DHT22_PORT, DHT22_PIN)));
    }
    return i;
}
```

Listing 3 – Extrait de code pour le DHT22 avec STM32

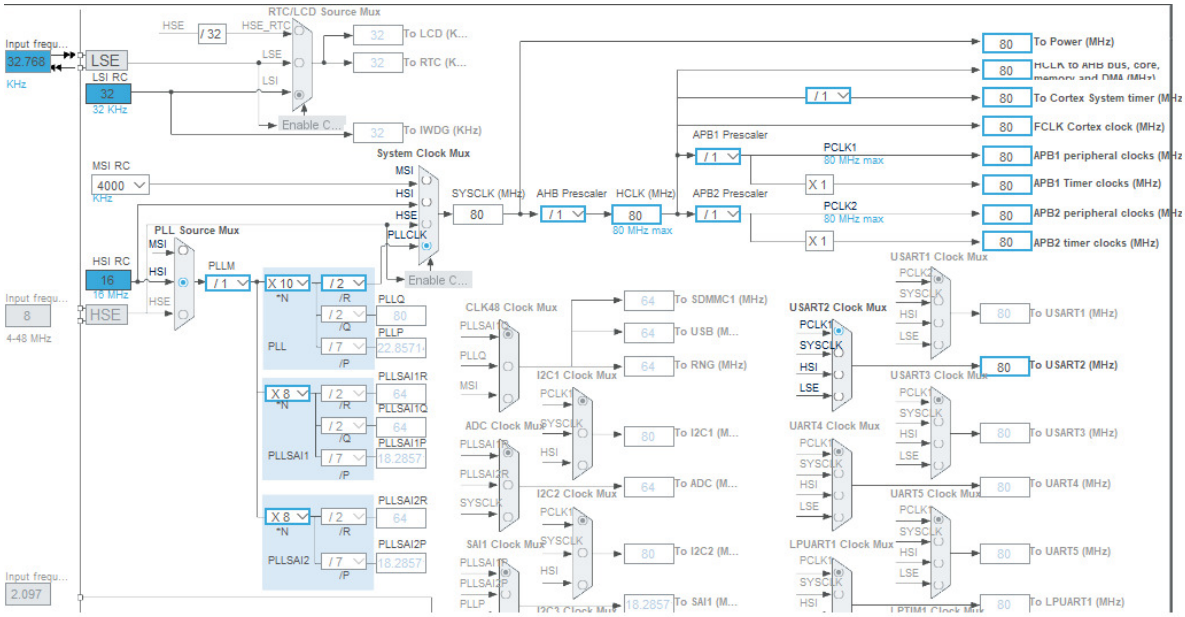

FIGURE I.10 – Configuration des timers pour le DHT22.

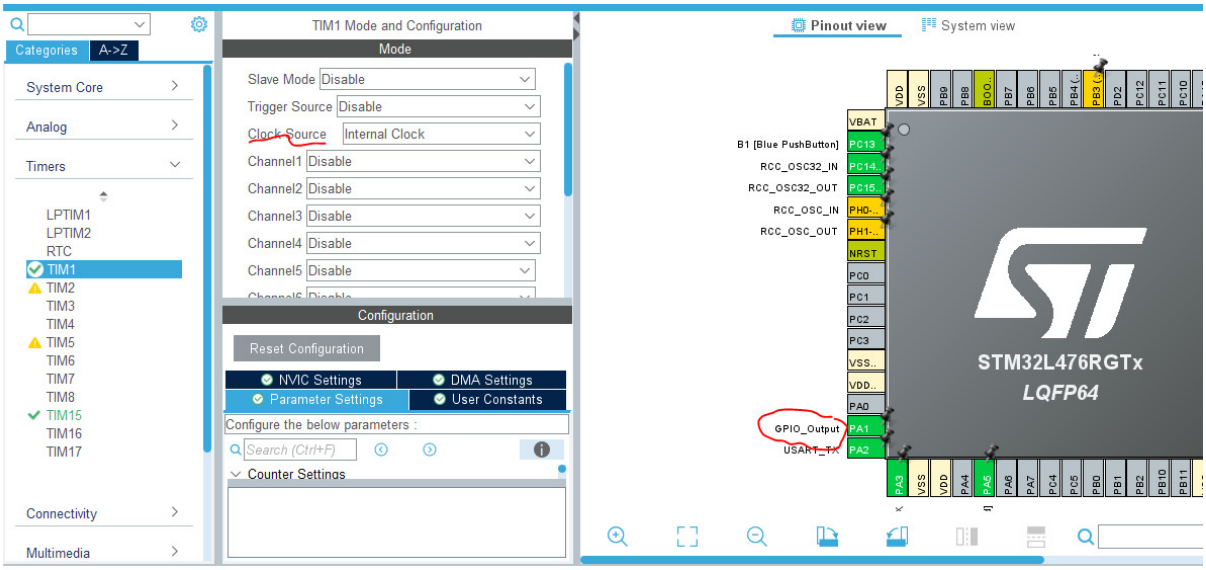

FIGURE I.11 – Mode et configuration du TIM1 pour le DHT22.

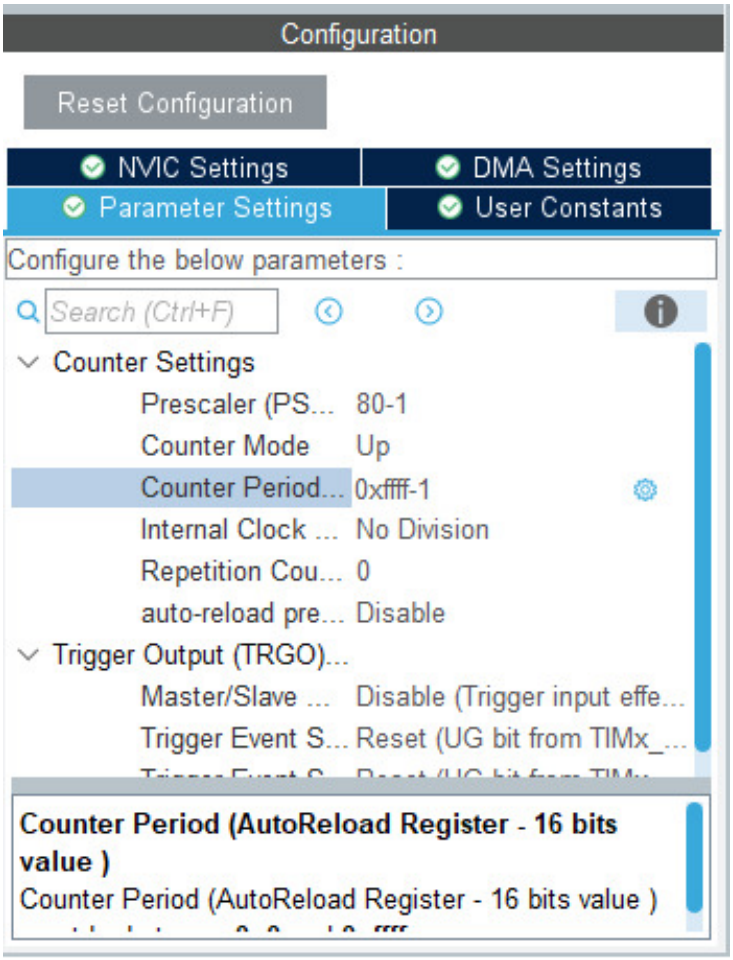

FIGURE I.12 – Zoom sur les paramètres de configuration du TIM1 pour le DHT22.

2.6 Applications

Le DHT22 est idéal pour la surveillance environnementale, les systèmes de contrôle climatique, et les projets de domotique. Sa précision et sa facilité d'intégration en font un choix populaire pour les amateurs et les professionnels.

2.7 Conclusion

Le capteur DHT22 est un outil précieux pour toute application nécessitant une mesure précise de la température et de l'humidité. Avec sa plage de mesure étendue et sa facilité d'utilisation, il reste un choix populaire pour les projets électroniques nécessitant des capteurs environnementaux fiables.

Pour plus de détails techniques, vous pouvez consulter la fiche technique du DHT22 disponible sur des sites comme SparkFun ou Adafruit.

3 Capteur HX711

3.1 Introduction

Le capteur HX711 est un convertisseur analogique-numérique (ADC) 24 bits conçu pour les applications de pesée et de contrôle industriel. Il est basé sur la technologie

brevetée d'Avia Semiconductor et est capable d'interfacer directement avec un capteur à pont.

3.2 Caractéristiques Techniques

- **Canaux d'entrée** : Deux canaux différentiels sélectionnables.
- **PGA** : Amplificateur de gain programmable avec des gains de 32, 64 et 128.
- **Régulateur de tension** : Régulateur de tension intégré pour l'alimentation du capteur et de l'ADC.
- **Horloge** : Horloge flexible pouvant provenir d'une source externe, d'un cristal ou de l'oscillateur interne.
- **Contrôle numérique** : Contrôle numérique simple par broches.
- **Consommation d'énergie** : $< 1.5\,\text{mA}$ en fonctionnement normal, $< 1\,\mu\text{A}$ en mode veille.
- **Plage de température de fonctionnement** : $-40\,^\circ\text{C}$ à $+85\,^\circ\text{C}$.
- **Tension d'alimentation** : $2.6\,\text{V}$ à $5.5\,\text{V}$.

3.3 Connexion et Utilisation

Le HX711 peut être connecté à un microcontrôleur via une interface série. Les broches principales du capteur sont :
- **AVDD** : Alimentation analogique (2.6 V à 5.5 V).
- **AGND** : Masse analogique.
- **PD_SCK** : Entrée d'horloge série et de contrôle de mise en veille.
- **DOUT** : Sortie de données série.

3.4 Exemple de Code

Voici un exemple de code pour interfacer le capteur HX711 avec un microcontrôleur STM32 :

```
#include "main.h"
#include "gpio.h"

// Definition des broches pour DOUT et SCK
#define DOUT_PIN GPIO_PIN_8
#define DOUT_PORT GPIOB
#define SCK_PIN GPIO_PIN_9
#define SCK_PORT GPIOB

// Fonction pour lire les donnees du HX711
uint32_t getHX711(void) {
    uint32_t data = 0;
    // Attendre que DOUT soit bas
    while (HAL_GPIO_ReadPin(DOUT_PORT, DOUT_PIN) == GPIO_PIN_SET);
    // Lire les 24 bits de donnees
    for (int8_t i = 0; i < 24; i++) {
        HAL_GPIO_WritePin(SCK_PORT, SCK_PIN, GPIO_PIN_SET);
        data = data << 1;
        HAL_GPIO_WritePin(SCK_PORT, SCK_PIN, GPIO_PIN_RESET);
```


```
        if (HAL_GPIO_ReadPin(DOUT_PORT, DOUT_PIN) == GPIO_PIN_SET)
            data++;
    }
    // Inverser les bits pour obtenir la valeur correcte
    data = data ^ 0x800000;
    return data;
}

// Fonction pour calculer le poids
int weigh(void) {
    int32_t total = 0;
    int32_t samples = 50;
    int milligram;
    float coefficient;
    // Lire plusieurs echantillons et calculer la moyenne
    for (uint16_t i = 0; i < samples; i++) {
        total += getHX711();
    }
    int32_t average = (int32_t)(total / samples);
    // Calculer le poids en milligrammes
    coefficient = knownOriginal / knownHX711;
    milligram = (int)((average - tare) * coefficient);
    return milligram;
}

int main(void) {
    // Initialisation de l'horloge du systeme
    SystemClock_Config();
    // Initialisation des GPIO
    MX_GPIO_Init();
    // Configurer les broches DOUT et SCK
    HAL_GPIO_WritePin(SCK_PORT, SCK_PIN, GPIO_PIN_SET);
    HAL_Delay(10);
    HAL_GPIO_WritePin(SCK_PORT, SCK_PIN, GPIO_PIN_RESET);
    HAL_Delay(10);
    // Boucle principale
    while (1) {
        int weight = weigh(); // en milligrammes
    }
}
```

Listing 4 – Exemple de code pour le HX711 avec STM32

3.5 Applications

Le capteur HX711 est idéal pour les applications de pesée et de contrôle industriel nécessitant une conversion analogique-numérique précise. Sa faible consommation d'énergie et sa capacité à interfacer directement avec un capteur à pont en font un choix populaire pour les applications nécessitant une autonomie prolongée.

3.6 Conclusion

Le capteur HX711 est un composant essentiel pour les applications de pesée et de contrôle industriel. Avec sa précision et sa faible consommation d'énergie, il est parfaitement adapté pour les applications nécessitant une conversion analogique-numérique fiable

et efficace.

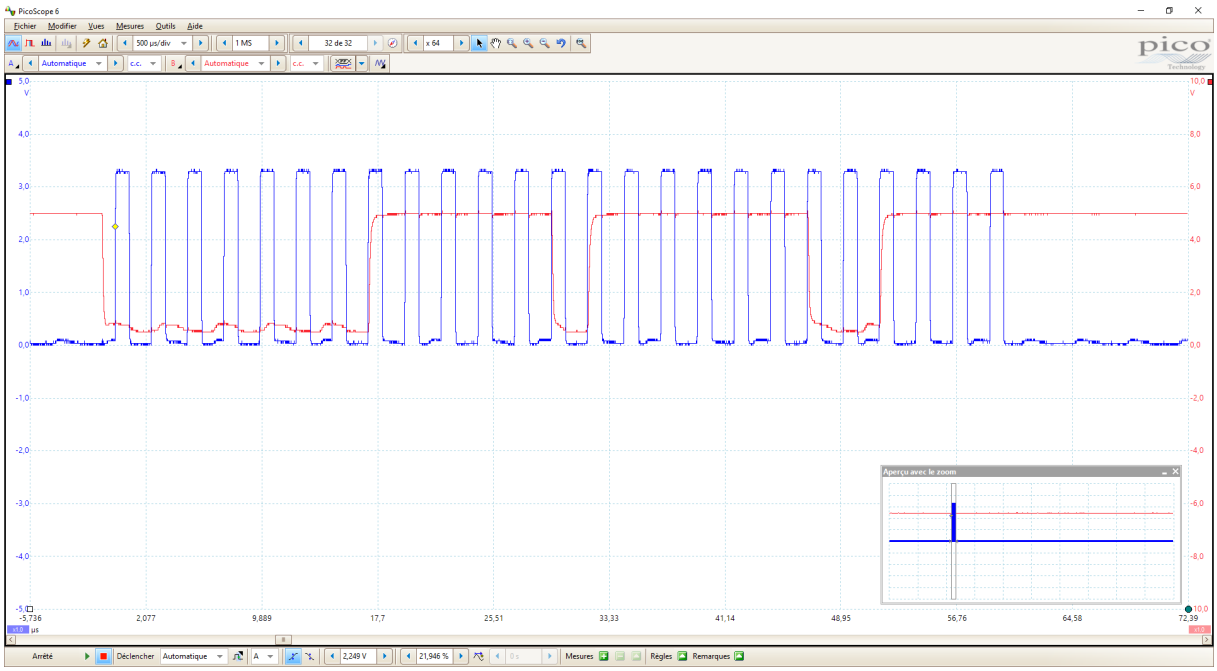

FIGURE I.13 – Trame du poids (bleu) et de l'horloge (rouge) du capteur HX711.

4 Capteur LoRa-E5

4.1 Introduction

Le module **LoRa-E5** repose sur le microcontrôleur **STM32WLE5JC**, intégrant une radio LoRa (SX126X) et un MCU ARM Cortex-M4. Il est destiné aux applications nécessitant une **communication longue portée**, **faible débit** et **très basse consommation énergétique**. Le module est conforme aux spécifications **LoRaWAN 1.0.3** et compatible avec les bandes ISM courantes (ex. 868 MHz pour l'Europe).

4.2 Caractéristiques Techniques

- **Bande de fréquence** : 150 – 960 MHz (support des bandes LoRaWAN régionales).
- **Portée** : Jusqu'à **15 km** (champ libre, dépend du facteur d'étalement).
- **Consommation** :
  - Mode veille : 2.1 µA (Deep Sleep).
  - Transmission à 14dBm : 43 mA.
  - Réception : 9 mA.
- **Protocoles supportés** : LoRaWAN Classe A/B/C, modulation LoRa et FSK.
- **Interface de commande** : UART (3.3V TTL).
- **Sécurité** : Chiffrement AES-128 matériel pour les données LoRaWAN.

4.3 Configuration STM32 pour le module LoRa-E5

Pour communiquer avec le module, le microcontrôleur STM32 utilise une interface UART configurée comme suit :

- **Baudrate** : 9600 bps.

— **Bits de données** : 8.
— **Bit de stop** : 1.
— **Parité** : Aucune.

Configuration typique dans STM32CubeMX :
— Activation de l'USART (ex : USART1).
— Activation des interruptions RX (réception asynchrone).
— Fonctions HAL utilisées : `HAL_UART_Transmit()` et `HAL_UART_Receive_IT()`.

Connexion physique :
— VCC → 3.3V.
— GND → Masse.
— TX (LoRa-E5) → RX (STM32).
— RX (LoRa-E5) → TX (STM32).

4.4 Protocole de Communication et Commande AT+OK

La communication avec le module se fait via l'envoi de **commandes AT** sur l'interface UART. Une commande basique pour tester la connectivité est `AT`, à laquelle le module répond `OK`.

Format des trames UART :
— **Trame envoyée (STM32 → LoRa-E5)** : `0x41 0x54 0x0D 0x0A` (`AT\r\n`).
— **Trame réponse (LoRa-E5 → STM32)** : `0x4F 0x4B 0x0D 0x0A` (`OK\r\n`).

Remarques importantes :
— Toute commande doit être terminée par `\r\n` (CR+LF).
— Temps de réponse typique : < 50 ms.
— Timeout UART recommandé : 100 ms.

4.5 Captures des Tests

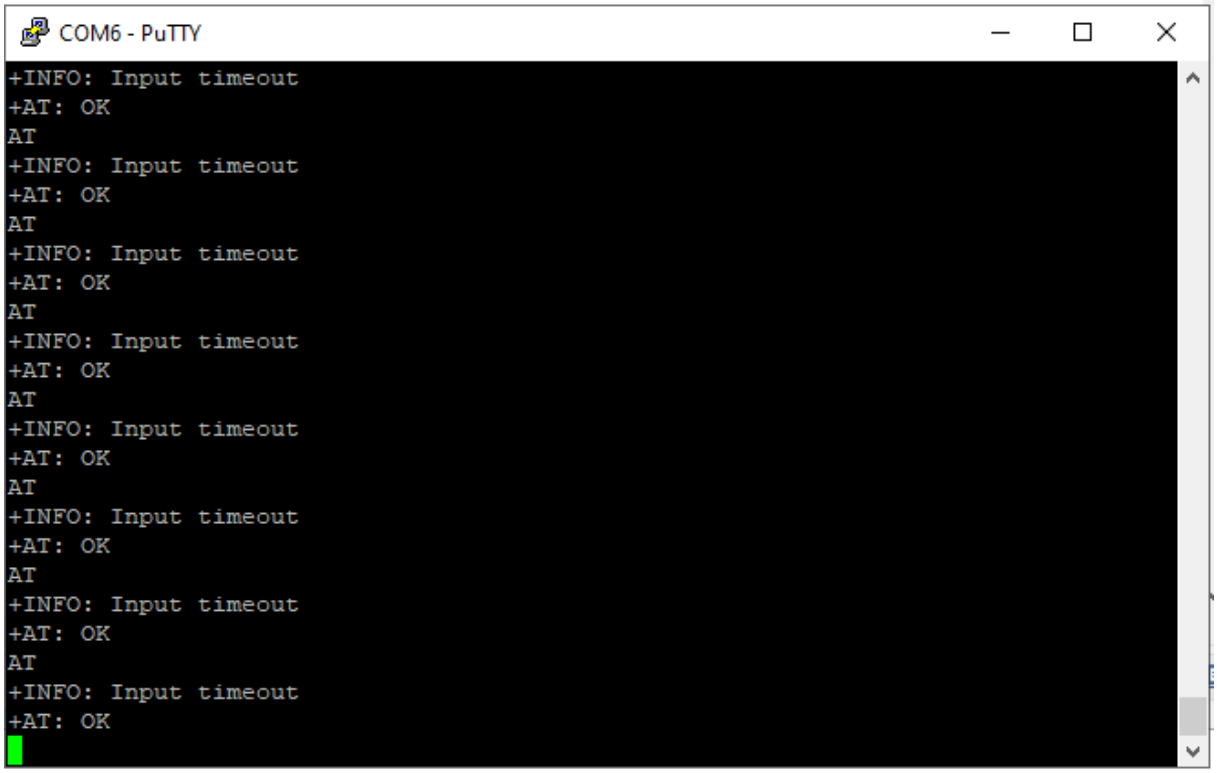

FIGURE I.14 – Réception de la réponse `OK` suite à la commande `AT` sur le terminal série.

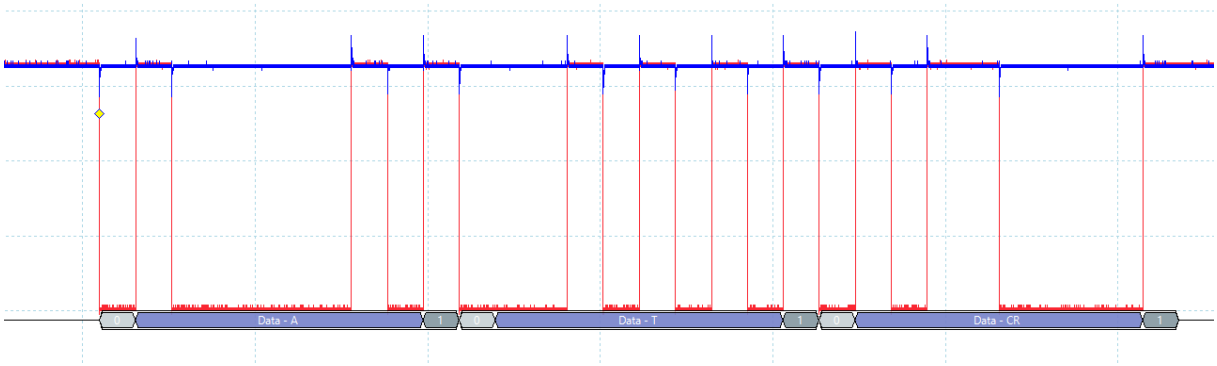

FIGURE I.15 – Trame UART envoyée au module LoRa-E5 (commande `AT`).

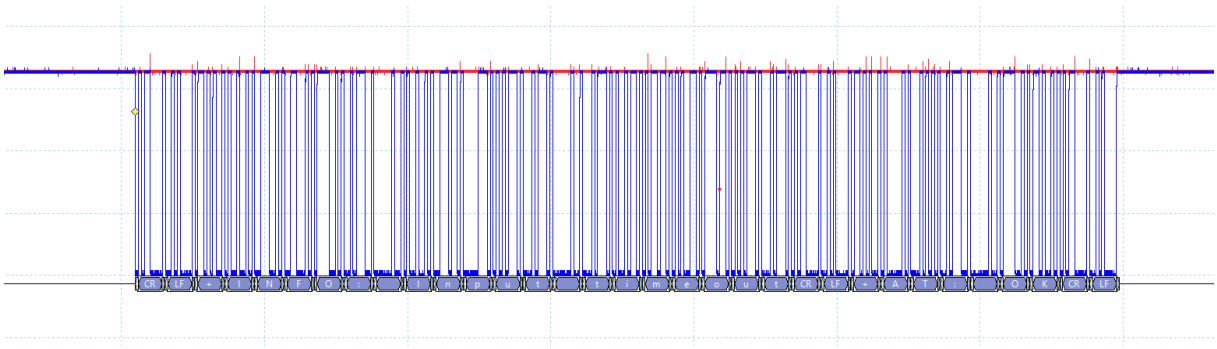

FIGURE I.16 – Trame UART reçue du module (réponse `OK`).

4.6 Analyse Technique de la Commande AT

La commande `AT` permet de :
— Vérifier que le module est bien **alimenté**.
— Tester la **connexion UART** (détection de croisement TX/RX).
— Confirmer que le module n'est pas **occupé** (mode transmission ou réception LoRa actif).

Débogage :
— **Aucune réponse** : vérifier alimentation, câblage, vitesse UART, terminaison `\r\n`.
— **Réponse ERROR** : syntaxe invalide ou module occupé.

4.7 Applications Typiques

Le module LoRa-E5 est particulièrement adapté pour :
— Réseaux de capteurs IoT longue portée (agriculture, smart city).
— Systèmes de télémétrie industriels.
— Suivi d'actifs (Asset Tracking).
— Monitoring environnemental (ex : température, humidité avec capteur DHT22).

4.8 Conclusion

Le **LoRa-E5** est un module clé pour l'IoT longue portée, combinant **performance radio**, **faible consommation** et **facilité de configuration via commandes AT**. La commande `AT/OK` permet une **validation rapide** de l'interface UART et du bon fonctionnement du module, préalable à toute communication LoRaWAN.

5 Exploitation des données

Les données récupérées des capteurs SHT31, DHT22 et HX711 sont ensuite traitées et analysées pour être intégrées dans le système de la ruche connectée.

II BE Ruche connectée - Partie Transmission

1 Configuration du Module LoRa-E5

Le module LoRa-E5 est configuré pour communiquer via l'interface UART du microcontrôleur STM32. Les paramètres de configuration incluent le baudrate, les bits de données, le bit de stop et la parité.

- **Baudrate** : 9600 bps.
- **Bits de données** : 8.
- **Bit de stop** : 1.
- **Parité** : Aucune.

Ces paramètres sont configurés dans STM32CubeMX pour assurer une communication fiable entre le microcontrôleur et le module LoRa-E5.

2 Initialisation et Configuration

L'initialisation du module LoRa-E5 et des capteurs est réalisée dans la fonction `initializeSensors(`
Cette fonction configure les capteurs et envoie les commandes initiales au module LoRa.

```
void initializeSensors(void) {
  HAL_TIM_Base_Start(&htim6); // D marrer le timer
  HX711_Start();

  // Initialiser l'afficheur LCD
  lcd_init(&hi2c1, &lcd);
  displayLCDMessage("Calibration", "Weight Sensor");
  HAL_Delay(3000);

  // D marrer l'auto-calibration
  calibrate(&knownHxVal, &tareVal);
  displayLCDMessage("Calibration", "Complete");
}
```

Listing 5 – Initialisation des capteurs et du module LoRa

3 Envoi de Commandes au Module LoRa

La fonction `sendLoRaCommand()` est utilisée pour envoyer des commandes au module LoRa via l'interface UART. Cette fonction prend en paramètre une chaîne de caractères représentant la commande à envoyer.

```
void sendLoRaCommand(const char* cmd) {
  snprintf(loraCmd, sizeof(loraCmd), "%s", cmd);
  HAL_UART_Transmit(&huart1, (uint8_t*)loraCmd, strlen(loraCmd), 1000);
  HAL_UART_Receive(&huart1, (uint8_t*)loraResp, sizeof(loraResp), 1000);
}
```

Listing 6 – Envoi de commandes au module LoRa

4 Traitement des Données des Capteurs

Les données des capteurs SHT31, DHT22 et HX711 sont traitées et envoyées via le module LoRa. Les fonctions `processSHT31Data()`, `processDHT22Data()` et `processHX711Data()`

sont utilisées pour lire les données des capteurs, les convertir en valeurs exploitables et les envoyer via le module LoRa.

4.1 Traitement des Données du Capteur SHT31

La fonction `processSHT31Data()` lit les données du capteur SHT31, les convertit en valeurs de température et d'humidité, et les envoie via le module LoRa.

```
void processSHT31Data(void) {
  // Envoyer la commande I2C pour configurer le capteur SHT31
  HAL_I2C_Master_Transmit(&hi2c1, (0x44 << 1), shtCmd, 2, 5000);
  HAL_Delay(50);

  // Lire les donn es du capteur SHT31
  HAL_I2C_Master_Receive(&hi2c1, (0x44 << 1), shtData, 6, 5000);
  HAL_Delay(50);

  // Convertir les donn es brutes en temp rature et humidit
  rawTemp = (shtData[0] << 8) | shtData[1];
  temp1 = -45 + 175 * (rawTemp / 65535.0);
  rawHum = (shtData[3] << 8) | shtData[4];
  hum1 = 100 * (rawHum / 65535.0);

  // Afficher les donn es sur l'afficheur LCD
  char dispTemp1[20], dispHum1[20];
  sprintf(dispTemp1, "Temp: %.2f C", temp1);
  sprintf(dispHum1, "Hum: %.2f %%", hum1);
  displayLCDMessage(dispTemp1, dispHum1);

  // Pr parer et envoyer le message LoRa
  char sensor1Id[3] = {"01"};
  int tempInt1 = (int)temp1;
  float tempDec1 = temp1 - tempInt1;
  int tempDecInt1 = (int)(tempDec1 * 100);
  int humInt1 = (int)hum1;
  float humDec1 = hum1 - humInt1;
  int humDecInt1 = (int)(humDec1 * 100);

  snprintf(loraMsg, sizeof(loraMsg), "AT+TEST=TXLRPKT \"%s%02d%02d%02d
   %02d\"\r\n", sensor1Id, tempInt1, tempDecInt1, humInt1, humDecInt1);
  HAL_UART_Transmit(&huart1, (uint8_t*)loraMsg, strlen(loraMsg), 1000);
  HAL_UART_Receive(&huart1, (uint8_t*)loraRetMsg, sizeof(loraRetMsg),
   1000);

  // Vider les buffers
  memset(loraMsg, 0, sizeof(loraMsg));
  memset(loraRetMsg, 0, sizeof(loraRetMsg));
}
```

Listing 7 – Traitement des données du capteur SHT31

4.2 Traitement des Données du Capteur DHT22

La fonction `processDHT22Data()` lit les données du capteur DHT22, les convertit en valeurs de température et d'humidité, et les envoie via le module LoRa.

```
void processDHT22Data(void) {
  DHT22_Get_Data(&temp2, &hum2);

  // Afficher les donn es sur l'afficheur LCD
  char dispTemp2[20], dispHum2[20];
  sprintf(dispTemp2, "Temp: %.1f C", temp2);
  sprintf(dispHum2, "Hum: %.1f %%", hum2);
  displayLCDMessage(dispTemp2, dispHum2);

  // Pr parer et envoyer le message LoRa
  char sensor3Id[3] = {"03"};
  int tempInt2 = (int)temp2;
  float tempDec2 = temp2 - tempInt2;
  int tempDecInt2 = (int)(tempDec2 * 10);
  int humInt2 = (int)hum2;
  float humDec2 = hum2 - humInt2;
  int humDecInt2 = (int)(humDec2 * 10);

  snprintf(loraMsg, sizeof(loraMsg), "AT+TEST=TXLRPKT \"%s%02d%01d%02d
   %01d\"\r\n", sensor3Id, tempInt2, tempDecInt2, humInt2, humDecInt2);
  HAL_UART_Transmit(&huart1, (uint8_t*)loraMsg, strlen(loraMsg), 1000);
  HAL_UART_Receive(&huart1, (uint8_t*)loraRetMsg, sizeof(loraRetMsg),
   1000);

  // Vider les buffers
  memset(loraMsg, 0, sizeof(loraMsg));
  memset(loraRetMsg, 0, sizeof(loraRetMsg));
}
```

Listing 8 – Traitement des données du capteur DHT22

4.3 Traitement des Données du Capteur HX711

La fonction `processHX711Data()` lit les données du capteur HX711, les convertit en valeurs de poids, et les envoie via le module LoRa.

```
void processHX711Data(void) {
  char sensor2Id[3] = {"02"};
  char loraWeight[20];

  weightVal = weigh(knownWeight, knownHxVal, tareVal);

  if (weightVal > 0) {
      sprintf(loraWeight, "%d", weightVal);

      // Construire et envoyer le message LoRa
      snprintf(loraMsg, sizeof(loraMsg), "AT+TEST=TXLRPKT \"%s%s\"\r\n",
    sensor2Id, loraWeight);
      HAL_UART_Transmit(&huart1, (uint8_t*)loraMsg, strlen(loraMsg),
   1000);
      HAL_UART_Receive(&huart1, (uint8_t*)loraRetMsg, sizeof(loraRetMsg)
   , 1000);

      // Vider les buffers
      memset(loraMsg, 0, sizeof(loraMsg));
      memset(loraRetMsg, 0, sizeof(loraRetMsg));
  }
```


```
}
```

Listing 9 – Traitement des données du capteur HX711

5 Affichage des Données sur l'Afficheur LCD

Les données des capteurs sont affichées sur un afficheur LCD. La fonction `displayLCDMessage()` est utilisée pour afficher les messages sur l'afficheur LCD.

```
void displayLCDMessage(const char* line1, const char* line2) {
  clearlcd();
  lcd_position(&hi2c1, 0, 0);
  lcd_print(&hi2c1, line1);
  lcd_position(&hi2c1, 0, 1);
  lcd_print(&hi2c1, line2);
}
```

Listing 10 – Affichage des données sur l'afficheur LCD

III BE Ruche connectée - Partie Acquisition

1 Initialisation des Périphériques

La fonction `initializePeripherals()` est utilisée pour initialiser les périphériques nécessaires, y compris l'afficheur LCD.

```
void initializePeripherals(void) {
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_USART1_UART_Init();
  MX_I2C1_Init();

  lcd_init(&hi2c1, &lcd);
  clearlcd();
}
```

Listing 11 – Initialisation des périphériques et de l'afficheur LCD

2 Envoi de Commandes au Module LoRa

La fonction `sendLoRaCommand()` est utilisée pour envoyer des commandes au module LoRa via l'interface UART. Cette fonction prend en paramètre une chaîne de caractères représentant la commande à envoyer.

```
void sendLoRaCommand(const char* cmd) {
  snprintf(loraCmd, sizeof(loraCmd), "%s", cmd);
  HAL_UART_Transmit(&huart1, (uint8_t*)loraCmd, strlen(loraCmd), 1000);
  HAL_UART_Receive(&huart1, (uint8_t*)loraResp, sizeof(loraResp), 1000);
}
```

Listing 12 – Envoi de commandes au module LoRa

3 Réception et Affichage des Données

La boucle principale du programme attend et récupère les messages reçus par le module LoRa. Les messages sont ensuite analysés pour déterminer leur contenu et afficher les informations appropriées sur l'afficheur LCD.

```
while (1) {
  // Attendre et r cup rer le message
  HAL_UART_Receive(&huart1, (uint8_t*)loraRetMsg, sizeof(loraRetMsg),
   1000);
  HAL_UART_Transmit(&huart2, (uint8_t*)loraRetMsg, strlen(loraRetMsg),
   1000);

  // V rifier si le message contient "+TEST: RX "
  char *start = strstr(loraRetMsg, "+TEST: RX \"");
  if (start != NULL) {
      start += 11; // Passer apr s "+TEST: RX \""
      displayData(start);
  }

  // Vider les buffers pour  viter  les donn es obsol tes
  clearBuffers();
```


```
}
```

Listing 13 – Réception et affichage des données

4 Affichage des Données sur l'Afficheur LCD

La fonction `displayData()` est utilisée pour afficher les données sur l'afficheur LCD en fonction de l'ID du message reçu.

```
void displayData(char *message) {
    char buff[20];

    // V rifier si le message provient du capteur de temp rature et d'
    humidit  externe (ID "01")
    if (strncmp(message, "01", 2) == 0) {
        clearlcd();
        lcd_position(&hi2c1, 0, 0);
        lcd_print(&hi2c1, "Temp & Humidity");
        lcd_position(&hi2c1, 0, 1);
        lcd_print(&hi2c1, "Outside");
        HAL_Delay(2000);
        clearlcd();

        // Extraire et afficher la temp rature
        snprintf(buff, sizeof(buff), "Temp: %c%c.%c%c C", message[2],
    message[3], message[4], message[5]);
        lcd_position(&hi2c1, 0, 0);
        lcd_print(&hi2c1, buff);
        memset(buff, 0, sizeof(buff));

        // Extraire et afficher l'humidit
        snprintf(buff, sizeof(buff), "Hum: %c%c.%c%c %%", message[6],
    message[7], message[8], message[9]);
        lcd_position(&hi2c1, 0, 1);
        lcd_print(&hi2c1, buff);
        memset(buff, 0, sizeof(buff));
    }

    // V rifier si le message provient du capteur de poids (ID "02")
    else if (strncmp(message, "02", 2) == 0) {
        weightMg = atoi(&message[2]); // Convertir la cha ne en entier
    (mg)
        weightG = weightMg / 1000.0; // Convertir en grammes

        clearlcd();
        lcd_position(&hi2c1, 2, 0);
        lcd_print(&hi2c1, "Current Weight");
        lcd_position(&hi2c1, 2, 1);
        lcd_print(&hi2c1, "in Hive");
        HAL_Delay(2000);
        clearlcd();

        lcd_position(&hi2c1, 0, 0);
        lcd_print(&hi2c1, "Hive Weight:");
        snprintf(buff, sizeof(buff), "%.2f g", weightG);
        lcd_position(&hi2c1, 0, 1);
        lcd_print(&hi2c1, buff);
```


```
45          memset(buff, 0, sizeof(buff));
46      }
47
48      // V rifier si le message provient du capteur de poids avec un
    autre format (ID "002")
49      else if (strncmp(message, "002", 3) == 0) {
50          weightMg = atoi(&message[3]);
51          weightG = weightMg / 1000.0;
52
53          clearlcd();
54          lcd_position(&hi2c1, 2, 0);
55          lcd_print(&hi2c1, "Current Weight");
56          lcd_position(&hi2c1, 2, 1);
57          lcd_print(&hi2c1, "in Hive");
58          HAL_Delay(2000);
59          clearlcd();
60
61          lcd_position(&hi2c1, 0, 0);
62          lcd_print(&hi2c1, "Hive Weight:");
63          snprintf(buff, sizeof(buff), "%.2f g", weightG);
64          lcd_position(&hi2c1, 0, 1);
65          lcd_print(&hi2c1, buff);
66          memset(buff, 0, sizeof(buff));
67      }
68
69      // V rifier si le message provient du capteur de temp rature et d'
    humidit  interne (ID "03")
70      else if (strncmp(message, "03", 2) == 0) {
71          clearlcd();
72          lcd_position(&hi2c1, 0, 0);
73          lcd_print(&hi2c1, "Temp & Humidity");
74          lcd_position(&hi2c1, 0, 1);
75          lcd_print(&hi2c1, "Inside");
76          HAL_Delay(2000);
77          clearlcd();
78
79          // Extraire et afficher la temp rature
80          snprintf(buff, sizeof(buff), "Temp: %c%c.%c C", message[2],
    message[3], message[4]);
81          lcd_position(&hi2c1, 0, 0);
82          lcd_print(&hi2c1, buff);
83          memset(buff, 0, sizeof(buff));
84
85          // Extraire et afficher l'humidit
86          snprintf(buff, sizeof(buff), "Hum: %c%c.%c %%", message[5],
    message[6], message[7]);
87          lcd_position(&hi2c1, 0, 1);
88          lcd_print(&hi2c1, buff);
89          memset(buff, 0, sizeof(buff));
90      }
91
92      // V rifier si la pr sence est d tect e (ID "04")
93      else if (strncmp(message, "04", 2) == 0) {
94          clearlcd();
95          lcd_position(&hi2c1, 2, 0);
96          lcd_print(&hi2c1, "PRESENCE");
97          lcd_position(&hi2c1, 1, 1);
```


```
98            lcd_print(&hi2c1, "DETECTED");
99        }
100 }
```

Listing 14 – Affichage des données sur l'afficheur LCD

5 Vider les Buffers

La fonction `clearBuffers()` est utilisée pour vider les buffers de commande et de réponse afin d'éviter les données obsolètes.

```
1 void clearBuffers(void) {
2   memset(loraMsg, 0, sizeof(loraMsg));
3   memset(loraRetMsg, 0, sizeof(loraRetMsg));
4 }
```

Listing 15 – Vider les buffers

IV Câblage du Projet

Le câblage du projet est crucial pour assurer une communication correcte entre les différents composants. Voici une description détaillée du câblage :

1 Connexions des Capteurs

1.1 Capteur SHT31

Le capteur SHT31 est connecté via l'interface I2C du microcontrôleur STM32. Les connexions sont les suivantes :

- **VCC** : 3.3V (alimentation).
- **GND** : Masse.
- **SDA** : Pin PB7 (I2C1_SDA).
- **SCL** : Pin PB6 (I2C1_SCL).

1.2 Capteur DHT22

Le capteur DHT22 est connecté via un seul fil de données au microcontrôleur STM32. Les connexions sont les suivantes :

- **VCC** : 3.3V (alimentation).
- **GND** : Masse.
- **DATA** : Pin PA0 (GPIO_PIN_0).

1.3 Capteur HX711

Le capteur HX711 est connecté via une interface série au microcontrôleur STM32. Les connexions sont les suivantes :

- **AVDD** : 3.3V (alimentation analogique).
- **AGND** : Masse analogique.
- **DOUT** : Pin PB8 (GPIO_PIN_8).
- **SCK** : Pin PB9 (GPIO_PIN_9).

2 Connexions du Module LoRa-E5

Le module LoRa-E5 est connecté via l'interface UART du microcontrôleur STM32. Les connexions sont les suivantes :

- **VCC** : 3.3V (alimentation).
- **GND** : Masse.
- **TX** : Pin PA2 (USART2_TX).
- **RX** : Pin PA3 (USART2_RX).

3 Connexions de l'Afficheur LCD

L'afficheur LCD est connecté via l'interface I2C du microcontrôleur STM32. Les connexions sont les suivantes :

- **VCC** : 3.3V (alimentation).
- **GND** : Masse.
- **SDA** : Pin PB7 (I2C1_SDA).

— **SCL** : Pin PB6 (I2C1_SCL).

V Conclusion

Ce projet a permis de mettre en œuvre une ruche connectée capable de mesurer et de transmettre des données de température, d'humidité et de poids en temps réel. Les capteurs SHT31, DHT22 et HX711 ont été utilisés pour l'acquisition des données, tandis que le module LoRa-E5 a permis la transmission des données à distance. Le microcontrôleur STM32 a été utilisé pour gérer l'acquisition des données, la communication avec les capteurs et l'affichage des informations sur un afficheur LCD.

Les résultats obtenus montrent que le système est capable de mesurer et de transmettre les données de manière fiable, ce qui ouvre la voie à des applications pratiques dans divers domaines tels que l'agriculture, la gestion de l'énergie et la surveillance environnementale.