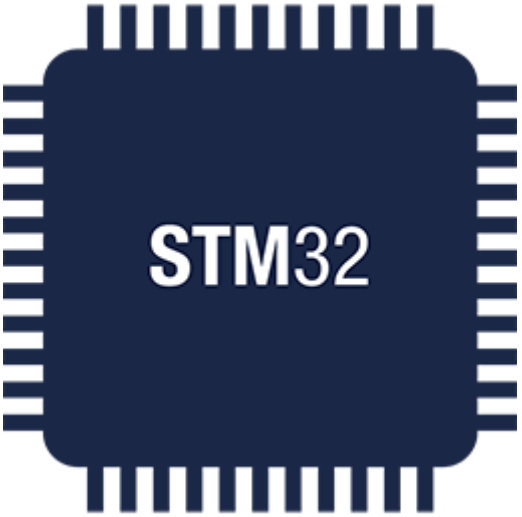
M1 EEA SYSTÈMES ET MICROSYSTÈMES EMBARQUÉS
2023-2024

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Compte Rendu Des Travaux Pratiques

UE : Réalisation des systèmes

TP : Projets de base



Présenté par : RUGHOONAUTH Pravin Raj
SELATNIA Mohamed Abdellah

Responsable : M. PERISSE

Table de matières

1. Introduction

L'objectif de ce projet est de former les étudiants à travailler avec la famille de cartes STM32 et le processeur ARM, en utilisant des capteurs pour répondre à des spécifications précises, tout en acquérant une meilleure compréhension des outils de développement fournis avec la carte NUCLEO STM32. Cette expérience offrira aux étudiants l'opportunité d'explorer des logiciels et du matériel couramment utilisés dans l'industrie.

Le projet sera divisé en deux parties distinctes : dans la première partie, nous aborderons le principe de fonctionnement ainsi que la mise en œuvre du capteur de température et d'humidité SHT31 en conjonction avec l'afficheur LCD fourni. La seconde partie se concentrera sur le processus de lecture des données du capteur DHT22, avec affichage des données correspondantes sur un autre afficheur LCD. L'objectif global est de comparer les deux protocoles de communication distincts utilisés pour chaque capteur, fournissant ainsi aux étudiants une vision approfondie des différentes approches techniques.

2. Matériels utilisés dans le projet

2.1 Carte Nucleo L476RG

La NUCLEO-L476RG (voir figure 1) représente une carte de développement STM32 dotée d'une architecture Cortex-M4 et d'un microcontrôleur STM32F410RB. Cette carte offre une plateforme flexible aux utilisateurs désireux d'explorer de nouvelles idées et de prototyper des projets avec la famille de microcontrôleurs STM32. Elle permet de choisir parmi différentes options de performances, de consommation d'énergie et de fonctionnalités.

Grâce à la prise en charge de la connectivité Arduino et aux headers ST Morpho, il est facile d'étendre les capacités de la plateforme de développement STM32 Nucleo en utilisant une vaste sélection de Shields spécialisés. La carte intègre un débogueur et un programmateur ST-LINK/V2-1, éliminant ainsi le besoin d'une sonde séparée. Elle est fournie avec la bibliothèque complète HAL STM32, une variété d'exemples de programmes et un accès direct aux ressources en ligne Mbed.

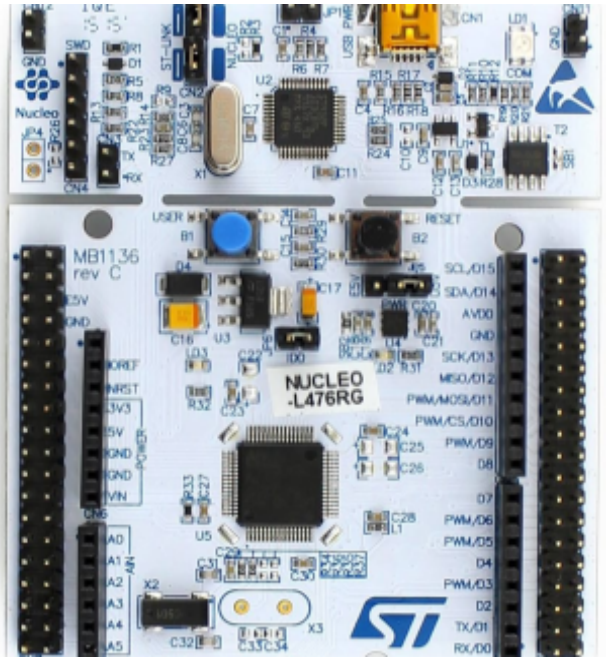


Figure 1 : Carte Nucleo L476RG

2.2 Capteur de température et d’humidité SHT31

Figure 2 : SHT31

Le capteur d'humidité et de température SHT31 repose sur une puce de capteur CMOSens, offrant ainsi une intelligence, une fiabilité et une précision élevées. Grâce à un traitement de signal amélioré, il propose deux adresses I2C distinctes et sélectionnables par l'utilisateur, ainsi que des vitesses de communication pouvant atteindre 1 MHz. Ce capteur garantit une précision typique de ± 1,5% pour l'humidité relative (HR) et de ± 0,2 °C pour la température. Avec ses dimensions compactes de 2,5 x 2,5 x 0,9 mm (longueur x largeur x hauteur) et une plage de tension d'alimentation allant de 2,4 à 5,5 V, il offre une solution polyvalente pour les applications nécessitant des mesures précises de l'humidité et de la température.

2.3 Capteur de température et d’humidité DHT22

Figure 3 : Capteur DHT22

Le capteur de température et d'humidité DHT22 communique avec un microcontrôleur via le protocole "one-wire". Il est pré-calibré et ne nécessite aucun composant supplémentaire pour son utilisation, car des résistances pull-up sont déjà intégrées. Ce module est compatible avec une plage de tension d'alimentation comprise entre 3,3 et 5,5 V, ce qui le rend adaptable à une variété d'applications sans nécessiter de modifications complexes du circuit.

2.4 Afficheur LCD I2C

Figure 4 : Afficheur LCD

L'afficheur LCD 2x16 caractères, rétro-éclairé, se connecte au bus I2C d'un microcontrôleur à l'aide d'un câble Grove et d'un Shield. Ce composant est compatible avec une plage de tension d'alimentation allant de 3,3 à 5,5 V, ce qui permet une utilisation flexible dans diverses applications. Grâce à son format compact et à son interface simplifiée via le bus I2C, cet afficheur offre une solution pratique pour l'affichage de données dans des projets électroniques.

2.5 Grove base shield

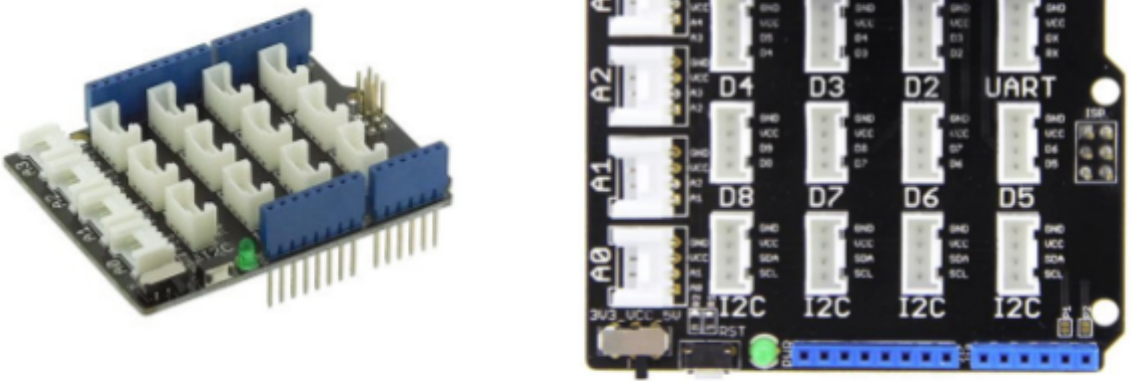


Figure 5 : Grove Base Shield

Le module Grove Base Shield est une carte d'interface conçue pour permettre une connexion simple, rapide et sans soudure des capteurs et des actionneurs Grove sur une carte compatible. Il est notamment compatible avec les cartes Arduino Uno, Leonardo, Seeeduino, ainsi qu'avec certaines cartes STM32 Nucleo 64. Ce Shield est équipé d'un sélecteur permettant de le rendre compatible avec des microcontrôleurs fonctionnant sous une tension de 3,3 V ou 5 V. Il dispose de 16 connecteurs à 4 broches, dont 4 sont des entrées analogiques, 7 sont des entrées/sorties logiques, 4 sont des interfaces

I2C et 1 est une interface UART. Cette polyvalence en termes de connectivité en fait un outil essentiel pour le prototypage et le développement de projets électroniques.

3. Les projets de base

3.1 SHT31 et LCD

Ce projet de base vise à mesurer la température et l'humidité en utilisant le capteur SHT31 dans n'importe quel environnement, puis à afficher les valeurs mesurées sur un afficheur LCD I2C.

3.1.1 Schéma de câblage

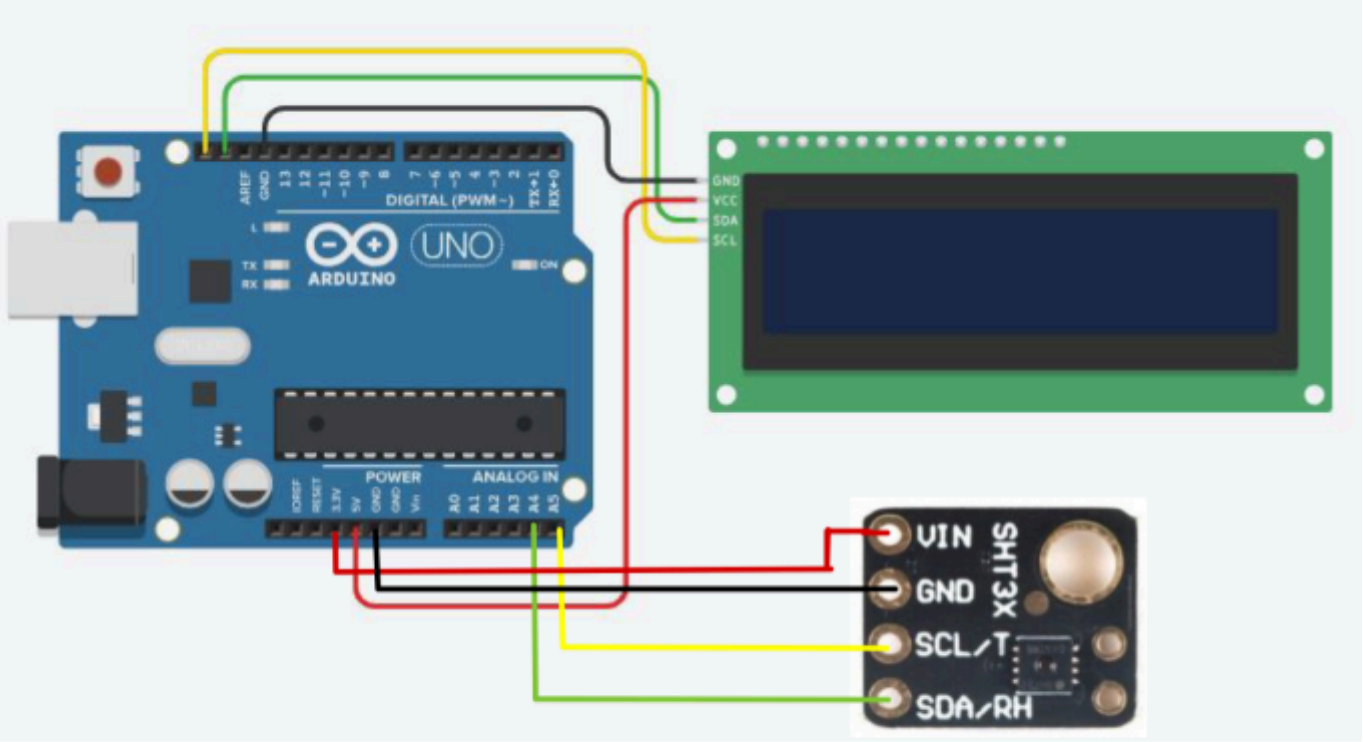


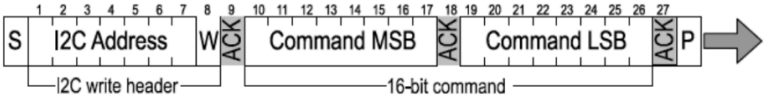
Figure 6 : Schéma de câblage SHT31 + LCD

Note : Dans le schéma de câblage présenté dans la Figure 6, nous avons utilisé la carte Arduino Uno pour représenter le module Grove Base Shield. Les broches A4 et A5 ainsi que les broches 14 et 15 sont respectivement désignées comme SDA et SCL.

3.1.2 Etude doc constructeur

- Capteur SHT31

Une fois que le capteur est connecté au shield et alimenté, il est prêt à recevoir des commandes. Le processus de transmission commence par une condition de départ (START) conformément aux normes I2C. L'en-tête d'écriture en I2C comprend donc une adresse de 7 bits (l'adresse du sht31 étant 0x44) suivie du bit d'écriture (0), puis est suivie par 16 bits de commande de mesure.

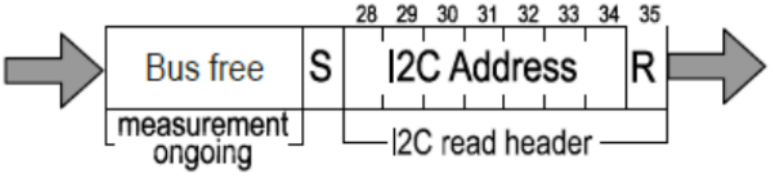


Condition		Hex. code	
Repeatability	Clock stretching	MSB	LSB
High	enabled	0x2C	06
Medium			0D
Low			10
High	disabled	0x24	00
Medium			0B
Low			16

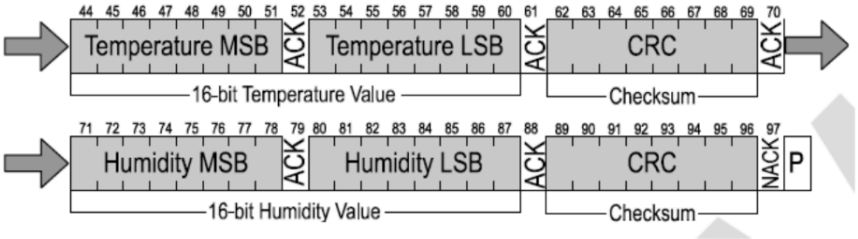
Pour notre cas, nous utilisons la commande 0x2416, ce qui correspond à une faible répétabilité et au désactivation du prolongement de l'horloge.

Après avoir reçu la commande, le capteur envoie un bit ACK (bit d'accusé de réception). Une fois ce bit envoyé, le capteur commence la mesure.

Pour lire les valeurs mesurées, nous envoyons la commande de lecture avec les 7 bits de l'adresse du capteur et un bit défini à 1 pour indiquer la lecture.



Ensuite, nous recevons les 16 bits de mesure de la température et les 16 bits de mesure de l'humidité, suivis à la fin de chaque mesure d'un CRC (Cyclic Redundancy Check).



Pour convertir les valeurs reçues en degrés Celsius et en pourcentage d'humidité relative (RH), nous devons utiliser 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):

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

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

- Carte Nucleo STM32L476RG

D'après les données recueillies sur le site MBED, il a été établi que le bus I2C est accessible à travers les broches PB8 et PB9, lesquelles correspondent respectivement aux signaux SCL et SDA. Donc, pour utiliser ce capteur avec CubeMX, nous devons activer les broches PB9 et PB8 comme SDA et SCL respectivement, afin de communiquer avec les ports I2C existants sur le shield Grove. L'ioc est montré dans la figure 7.

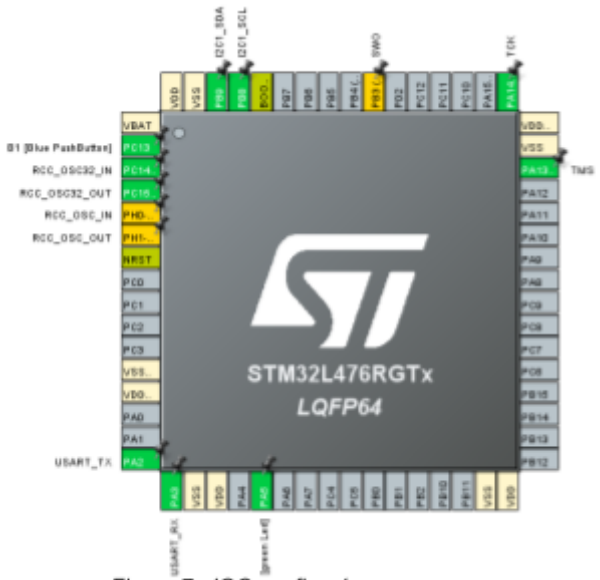


Figure 7 : IOC configuré

3.1.3 Programmation

Après avoir réalisé le câblage nécessaire, nous avons utilisé le logiciel STM32CubeMX pour configurer les broches requises pour notre projet et générer le code en langage C. Ensuite, nous avons apporté quelques modifications au programme généré en ajoutant des lignes de code à la boucle principale `while` montré dans la figure 8. Ces lignes permettent de lire les valeurs reçues sur les broches configurées et de les convertir en utilisant une équation de conversion trouvée dans la datasheet du

capteur. Enfin, nous avons affiché la valeur finale sur l'écran LCD selon les spécifications du programme ci-dessous. Le reste du code est disponible dans notre dépôt Github.

```
while (1)
{
    // Tableau pour stocker les commandes et les données
    buf[0] = CAPTEUR_CMD_MSB; // Octet de commande du poids fort
    buf[1] = CAPTEUR_CMD_LSB; // Octet de commande de poids faible

    // Transmission des commandes au capteur via le bus I2C
    ret = HAL_I2C_Master_Transmit( &hi2c1, CAPTEUR_ADRS, buf, 2, HAL_MAX_DELAY);
    if ( ret != HAL_OK)
    {
        // Si la transmission échoue, affiche un message d'erreur
        strcpy((char*)buf, "erreur_T!!\r\n");
    }
        else
    {

            // Réception des données du capteur via le bus I2C
    ret = HAL_I2C_Master_Receive( &hi2c1, CAPTEUR_ADRS, buf, 6, HAL_MAX_DELAY);
            if ( ret != HAL_OK)
            {
                strcpy((char*)buf, "erreur_R!!\r\n");
            }
    else
    {
        // Conversion des données de température
        valeur =   buf[1] | buf[0] << 8; // Combine les octets de données
        temp = -45 + 175 * ( (float)valeur / 65535); // convertit en degrés Celsius

        partieEntiere = (int) temp;
        partieDecimal = temp;
        partieDecimal *= 100;
        partieDecimal = partieDecimal - (partieEntiere * 100);

        // Conversion des données d'humidité
        valeur = buf[4] | buf[3] << 8;
        umid =  100 *( (float)valeur / 65535);

        //affichage lcd
        sprintf( (char*)buf, "%u.%u C ; %u %", (unsigned int) partieEntiere,(unsigned int) partieDecimal,(unsigned int) umid  );
        lcd_position(&hi2c1,0,0);
        lcd_print(&hi2c1,"Temp :  ");
        lcd_position(&hi2c1,7,0);
        lcd_print(&hi2c1,buf);
        lcd_position(&hi2c1,0,1);
        lcd_print(&hi2c1,"Hum  : ");
        lcd_position(&hi2c1,7,1);
        lcd_print(&hi2c1,&buf[10]);
        lcd_position(&hi2c1,11,1);
        lcd_print(&hi2c1,"%");


    }

    }

    HAL_UART_Transmit(&huart2, buf, strlen((char*)buf), HAL_MAX_DELAY);
    HAL_Delay(1000);
```

Figure 8 : La boucle “while”

3.1.4 Réalisation pratique et manipulation

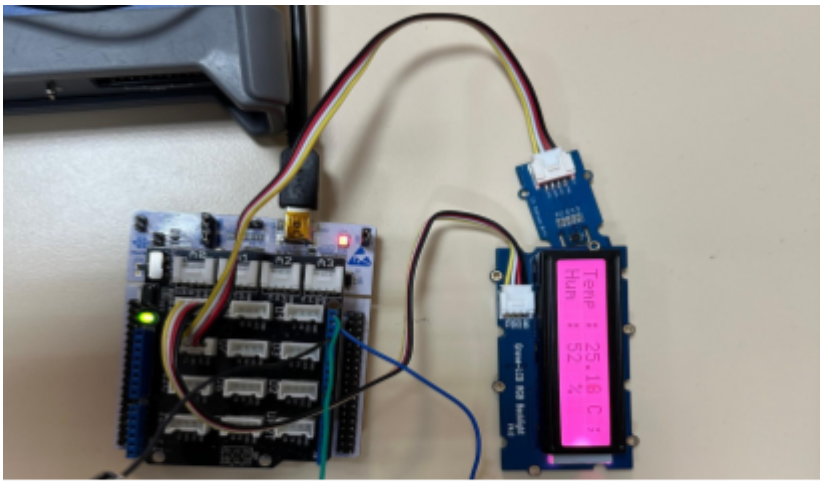


Figure 9 : Température et humidité affichées sur la LCD

Après avoir réalisé le câblage et écrit le code nécessaire, nous avons téléversé notre programme sur la carte pour mettre en marche notre projet. Un exemple des résultats obtenus est présenté sur la Figure 9 ci-dessus.

3.1.5 Visualisation : Picoscope

Dans cette phase, nous avons utilisé le PicoScope pour valider les résultats obtenus avec le capteur SHT31 et confirmer les informations I2C fournies par la datasheet. Des exemples des résultats du picoscope sont présentés sur la Figure 10, avec en bleu SCL et SDA en rouge.

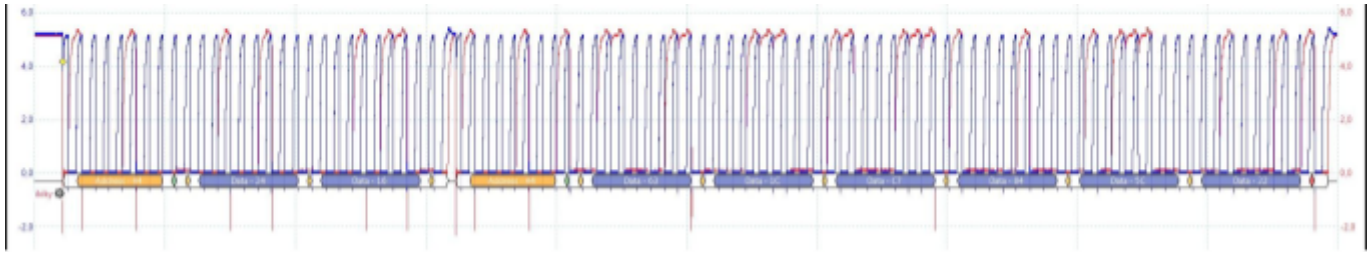


Figure 10 : SCL et SDA visualisés via le picoscope

Nous voyons bien l'adresse qui est de 0x44 ainsi que les commandes que nous avions choisi d'envoyer : 0x2416

Mesure Température

Donnés Température en Hexadécimal : 0x631C ——-----> En décimal : 25372

En utilisant la formule dans la datasheet : Température (°C) = -45 + 175*(25372/$(2^{16} - 1)$) = 24,75 °C

Mesure Humidité

Donnés Humidité en Hexadécimal : 0x845C ——-----> En décimal : 33884

En utilisant la formule dans la datasheet : Humidité (%) = (33884/$(2^{16} - 1)$)*100 = 51,70 %

Nous pouvons conclure que les données obtenues via le PicoScope sont conformes à ce qui est affiché sur l'écran LCD (Figure 9).

3.2 DHT22 et LCD

Ce TP de base vise à mesurer la température et l'humidité à l'aide du capteur DHT22 dans une plage de température possible de -40 à 80°C, puis à afficher les valeurs mesurées sur un afficheur LCD I2C.

3.2.1 Schéma de câblage

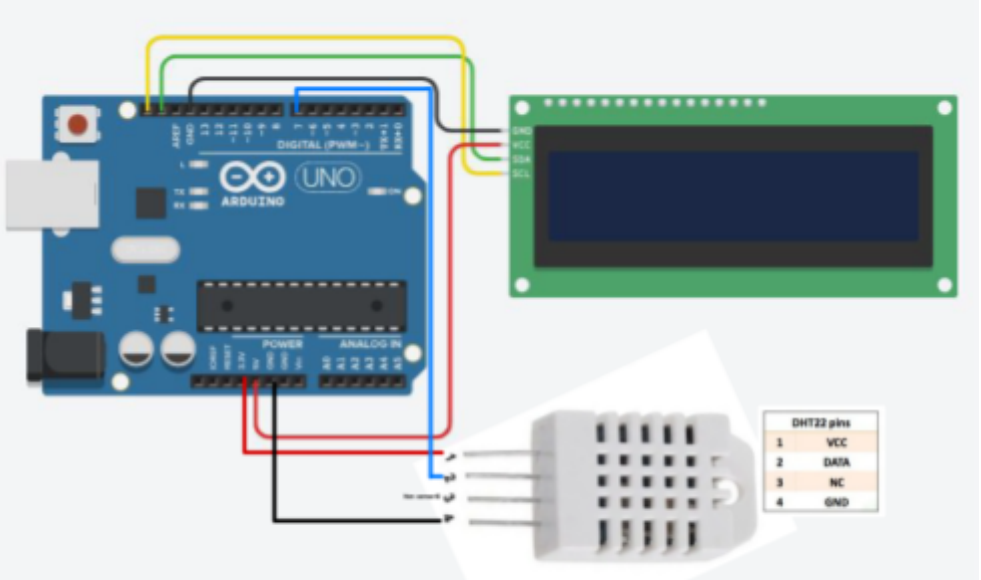


Figure 11 : Schéma de câblage DHT22 et LCD

Note : Dans le schéma de câblage présenté dans la Figure 11, nous avons utilisé la carte Arduino Uno pour représenter le module Grove Base Shield.

3.2.2 Etude doc constructeur

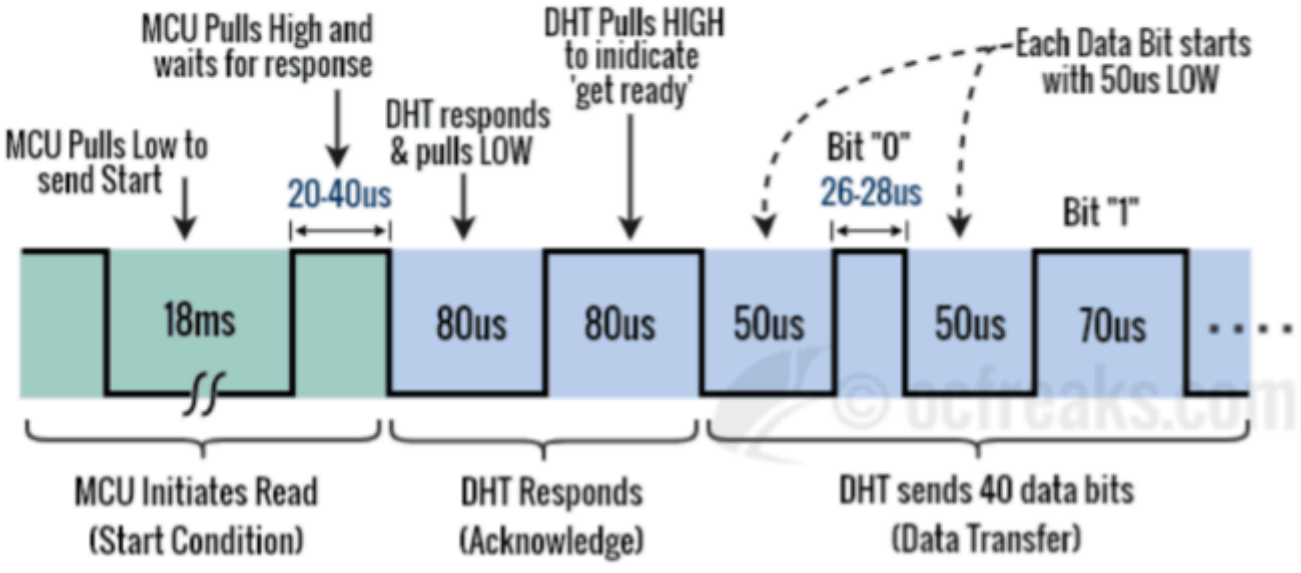


Figure 12: Protocole DHT22

Processus de communication du DHT22 de la figure 12 :

- Lorsque la ligne est au repos, le microcontrôleur la ramène à un niveau BAS pendant 18 ms.
- Ensuite, le MCU la tire HAUT pendant environ 20 à 40 µs.
- Le DHT22 détectera cela comme un DÉBUT émanant du MCU et répondra en tirant la ligne BAS pendant 80 µs.
- Ensuite, le DHT22 la tirera HAUT pendant 80 µs, ce qui indique qu'il est prêt à envoyer des données ou à "se préparer".
- Ensuite, il enverra 40 bits de données. Chaque bit commence par un niveau BAS de 50 µs suivi de 26 à 28 µs pour un "0" ou de 70 µs pour un "1".
- Après la fin de la communication, la ligne est tirée HAUT par la résistance de pull-up et entre dans l'état de repos.

Nous pouvons voir le format des données du capteur DHT22 dans la figure 13.

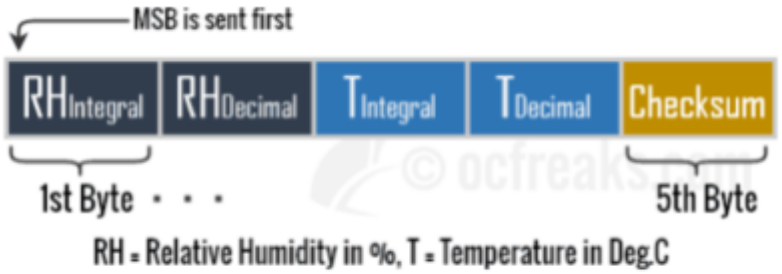


Figure 13 : Format de donnés du DHT22

3.2.3 Programmation

La fonction “delay” en microsecondes que nous avons développée s'avère particulièrement utile pour ce projet, car les durées des états haut et bas du capteur DHT22 sont de l'ordre de grandeur de cette échelle de temps. Elle est illustrée dans la figure 14.

```
//Fonction pour le timer en microseconde
void delay_us (uint16_t us)
{
    __HAL_TIM_SET_COUNTER(&htim6,0);  // initialiser le timer a 0
    while (__HAL_TIM_GET_COUNTER(&htim6)<us);  /*attendre que le compteur atteigne la
                                                valeur desirée mise en parametre*/
}
```

Figure 14 : Fonction delay en microsecondes

Les deux fonctions de la figure 15 permettent au protocole "1-wire" de passer entre les modes d'entrée et de sortie. Elles sont appelées dans la boucle principale.

```c
// Initialise la broche GPIO en mode sortie
void GPIO_INIT_OUTPUT(void){
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
    GPIO_InitStruct.Pin = GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

// Initialise la broche GPIO en mode entrée
void GPIO_INIT_INPUT(void){
        GPIO_InitTypeDef GPIO_InitStruct = {0};
        GPIO_InitStruct.Pin = GPIO_PIN_3;
        GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
```

Figure 15 : Mise en entrée et en sortie d'une broche

Quelques autres fonctions importantes sont illustrées dans la figure 16. Le reste du code est disponible dans notre dépôt Github.

```c
// Démarre la communication avec le capteur DHT22
id start_DHT22(void){
  GPIO_INIT_OUTPUT();
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
  HAL_Delay(1);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
  delay_us(35);
  GPIO_INIT_INPUT();
```

```c
// Vérifie la réponse du capteur DHT22
t8_t DHT22_Verification_Reponse(void){
 GPIO_INIT_INPUT();
 uint8_t Reponse = 0;
 delay_us(30);
 if (!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_3)))
 {
     delay_us(80);
         if ((HAL_GPIO_ReadPin (GPIOB, GPIO_PIN_3))) Reponse=1;
         else Reponse= -1;
 }
 while((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_3)))
         return Reponse;
```

```c
// Lit les données du capteur DHT22
uint8_t Lecture_DHT22(void){
    uint8_t octet_mesure,j;
    for(j=0;j<8;j++)
    {
        while (!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_3))){}
        delay_us(40);
        if (!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_3)))
        {
            octet_mesure &= ~(1<<(7-j));
        }
        else
        {
            octet_mesure |= (1<<(7-j));
        }
        while((HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_3))){}
    }
    return octet_mesure;
}
```

Figure 16 : Fonctions importantes

3.2.4 Réalisation pratique et manipulation

Après avoir réalisé le câblage et écrit le code nécessaire, nous avons téléversé notre programme sur la carte pour mettre en marche notre projet. Un exemple des résultats obtenus est présenté sur la Figure 17 ci-dessous.

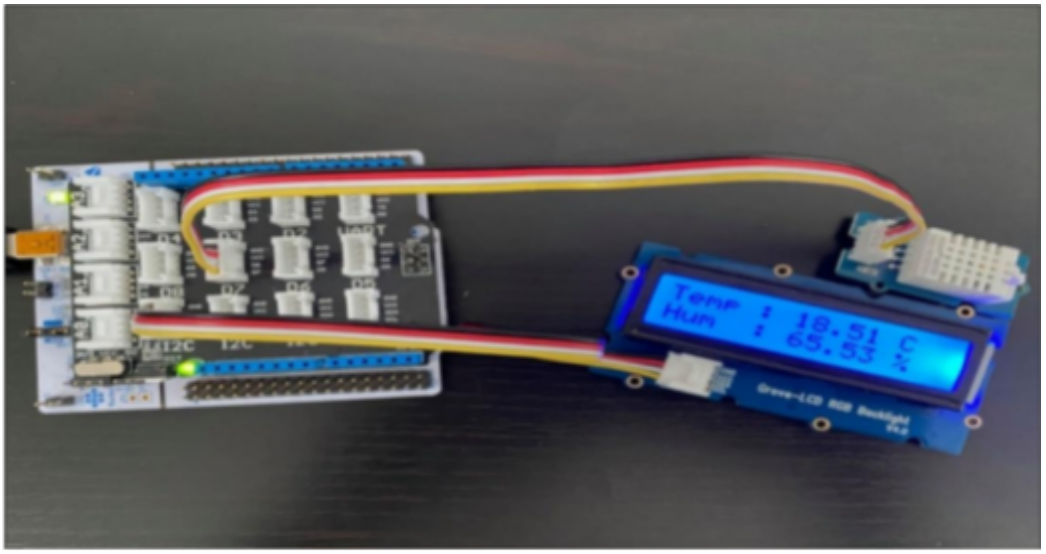


Figure 17 : Affichage de la température et de l'humidité

3.2.5 Visualisation : Picoscope

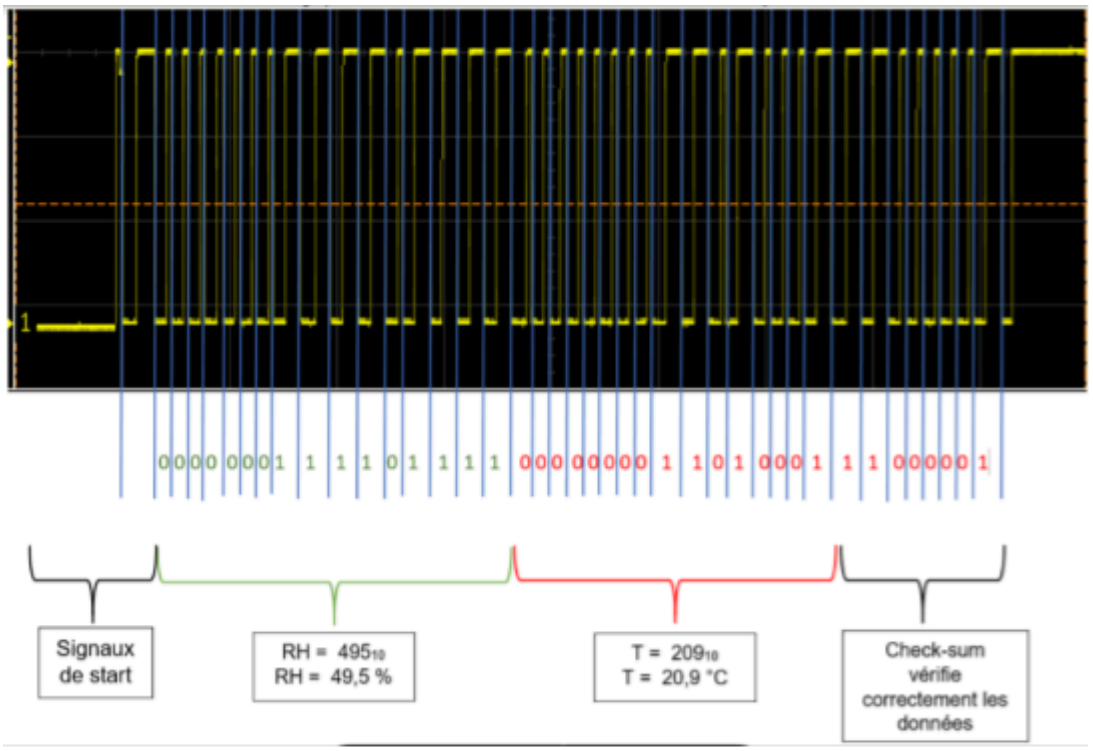


Figure 18 : Visualisation de donnée

4. Conclusion

Ce TP nous a offert l'opportunité de nous familiariser avec le microcontrôleur STM32 ainsi qu'avec les environnements de développement CubeIDE et CubeMX. De plus, nous avons étudié deux protocoles de communication : l'I2C et le 1-Wire.