
MASTER 1 SME

RAPPORT

# TP de BASE

Réalisé par :
BOUDOUNET Cécile
DOUKI Thiziri

*Encadré par :*
Mr. PERISSE Thierry

Année universitaire 2020-2021

# BE - TP de base

Cécile Boudounet - Thiziri Douki

April 2021

# Table des matières

# Introduction

Les TPs de base ont pour but de nous faire prendre en main à la fois les logiciels et le matériel nécessaire aux déroulements du BE.

A la fin de ces TPs, nous devons savoir utiliser CUBE-MX et TrueStudio et manipuler un microcontrôleur. Nous devons aussi être capable d'utiliser des capteurs fonctionnant avec différents protocoles.

Plus concrètement, nous devons à la fin de ces TPs pouvoir acquérir la température et le taux d'humidité d'une pièce à partir de deux capteurs et les afficher sur un écran LCD.

# Chapitre 1

## 1.1 Ecran LCD

### 1.1.1 Le protocole I2C :

Le bus I2C ( Inter Integrated Circuit ) a été développé pour permettre de relier facilement à un microprocesseur les différents circuits d'un téléviseur moderne.

C'est un protocole de communication série, les données sont donc transférées bit par bit le long d'un seul fil.

### 1.1.2 Caractèristiques :

Le bus I2C permet de faire communiquer entre eux des composants électroniques très divers grace à seulement trois fils :

— Un signal de donnée ( SDA ).
— Un signal d'horloge ( SCL ).
— Un signal de référence électrique ( Masse ).

### 1.1.3 Fonctionnement :

Avec I2C, les données sont transférées dans des messages. Les messages sont divisés en trames de données. Chaque message a une trame d'adresse qui contient l'adresse binaire de l'esclave, et une ou plusieurs trames de données qui contiennent les données en cours de transmission. Le message comprend également des conditions de démarrage et d'arrêt, des bits de lecture / écriture et des bits ACK / NACK entre chaque trame de données.

— **Condition de démarrage :** La ligne SDA passe d'un niveau de tension élevé à un niveau de tension bas avant que la ligne SCL ne passe du niveau haut au niveau

bas.

— **Condition d'arrêt :** La ligne SDA passe d'un niveau de tension bas à un niveau de tension élevé après que la ligne SCL passe du niveau bas au niveau haut.

— **Trame d'adresse :** Une séquence de 7 ou 10 bits unique à chaque esclave qui identifie l'esclave lorsque le maître veut communiquer avec lui.

— **Bit lecture / écriture :** Un seul bit spécifiant si le maître envoie des données à l'esclave (niveau de tension bas) ou lui demande des données (niveau de tension haut)

— **Bit ACK/NACK :** Chaque trame d'un message est suivie d'un bit d'acquittement/non-acquittement. Si une trame d'adresse ou une trame de données a été reçue avec succès, un bit ACK est renvoyé à l'expéditeur depuis le récepteur.

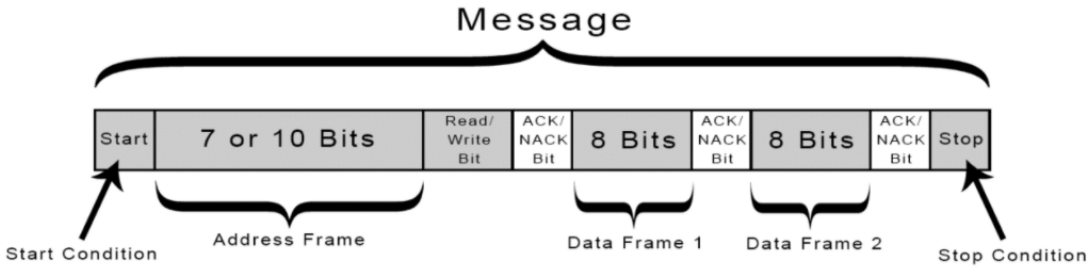
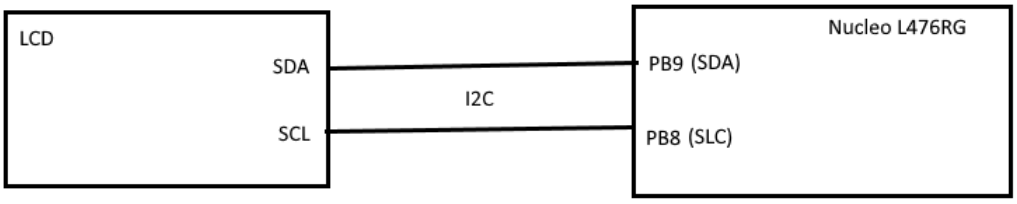



FIGURE 1.1 – Trame de données

### 1.1.4 Schéma de cablâge




### 1.1.5 Programmation

Pour utiliser l'écran LCD, nous avions à notre disposition une bibliothèque contenant des fonctions pour effacer l'écran, écrire sur l'écran, positionner le curseur...
Le LCD nous servira plus tard dans le TP pour afficher les températures et taux d'humidité relevés par les capteurs.

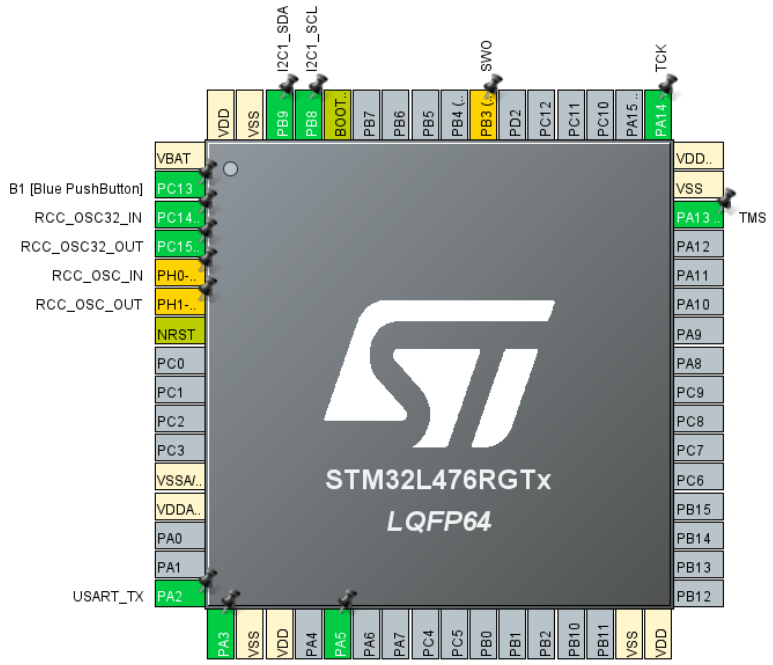
# Chapitre 2

## 2.1 Capteur SHT31

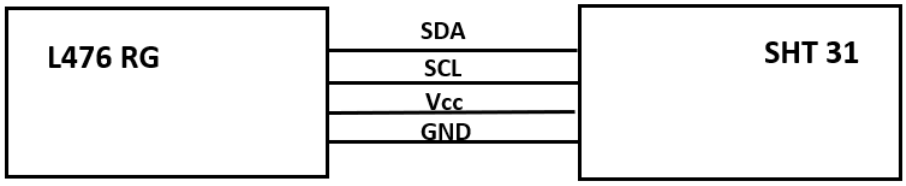
Le SHT31 est un capteur de température et d'humitidité qui utilise le protocole **I2C** pour la communication avec les microcontroleurs.

### 2.1.1 Schéma de cablâge

Pour relier le capteur et le microcontroleur, nous avons utilisé la PIN 8 du GPIOB comme SCL et la PIN 9 du GPIOB comme SDA :




Le schéma de cablage correspondant est le suivant :




### 2.1.2 Programmation

Nous avons utilisé les bibliothéques nécessaires pour l(utilisation d SHT31 et nous avons activé l' I2C1 sur CUBEMX. On utilise les lignes de code suisvantes pour lire les données du capteur depuis le registre puis calculer la température et l'humidité.

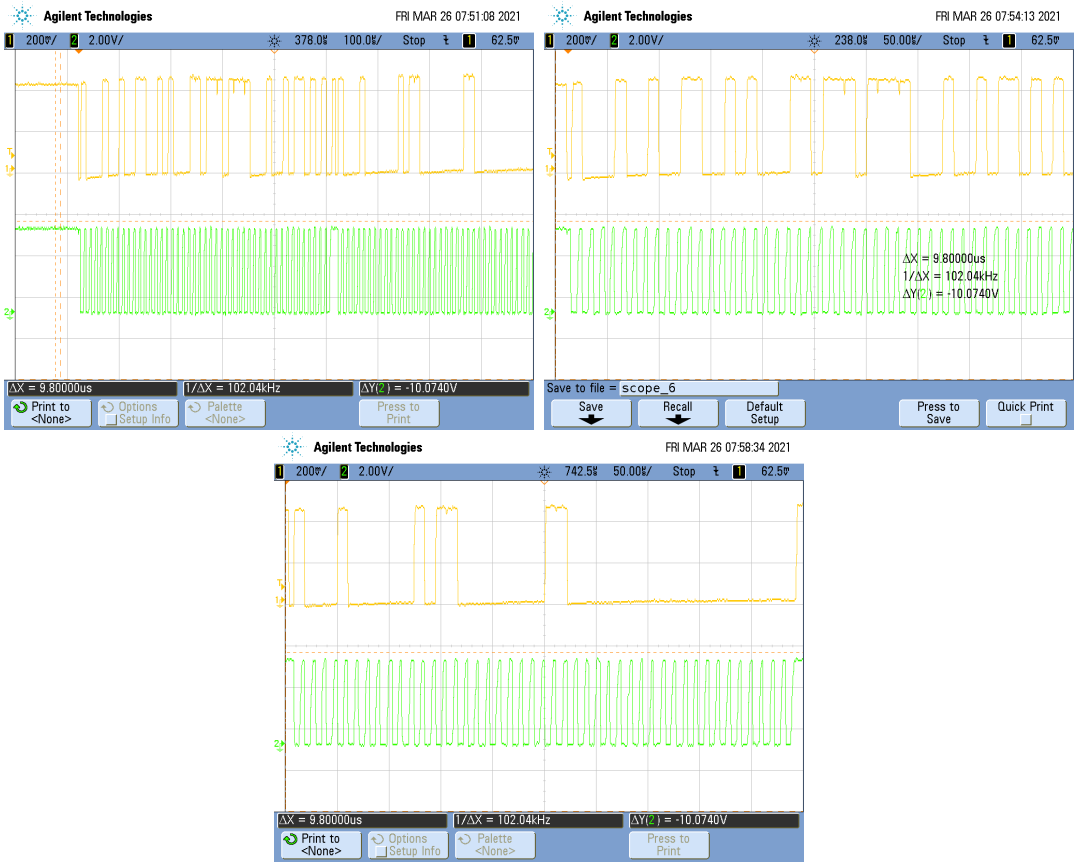
```
HAL_I2C_Master_Receive(&hi2c1,SHT31_ADDR | 0x01,data,4,50);
val = data[0] << 8 | data[1];
valH = data[0]<<8 | data[3];
temp = -45 +175 * ((float)val/65535);
hum = 100 * ((float)valH/65535);
```

### 2.1.3 Trame observée :

Nous avons observer la trame de données sur l'oscilloscope et nous avons trouvé le résultat suivant :

## 2.2 SHT31 + LCD :

Maintenant nous utilisont l'ecran LCD dans le but d'afficher la température ainsi que l'humidité. On trouvera ci dessous le shéma de cablage et l'output :



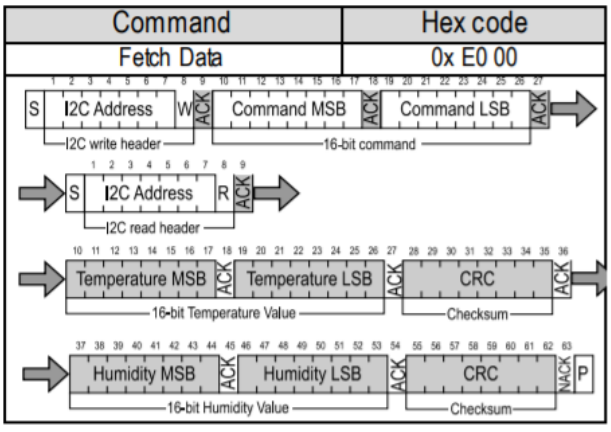
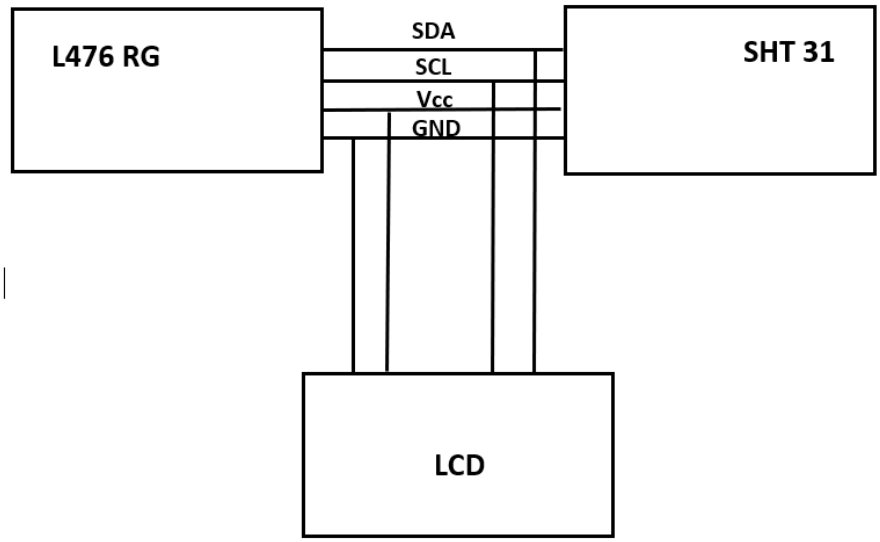




FIGURE 2.1 – Trame de données du SHT31



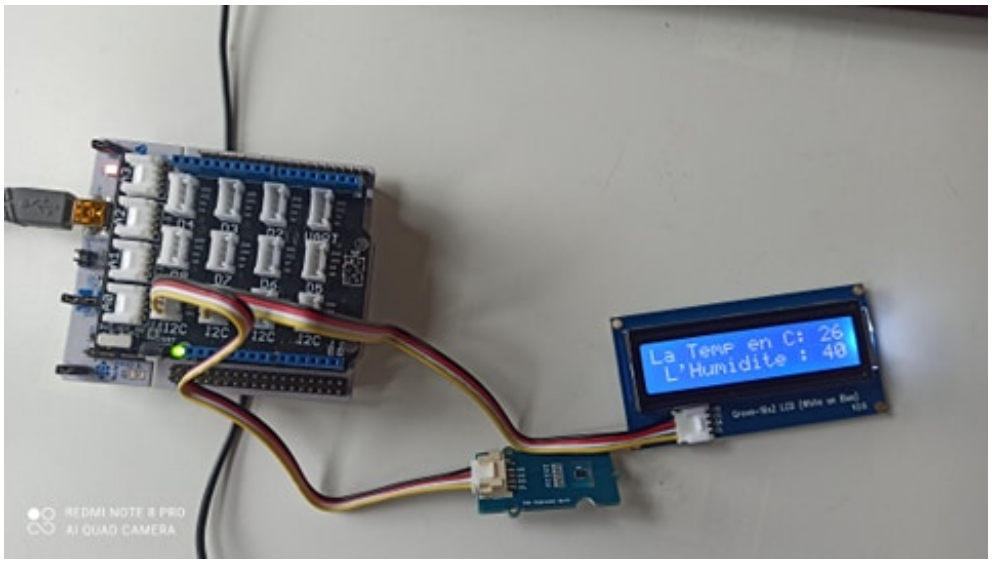



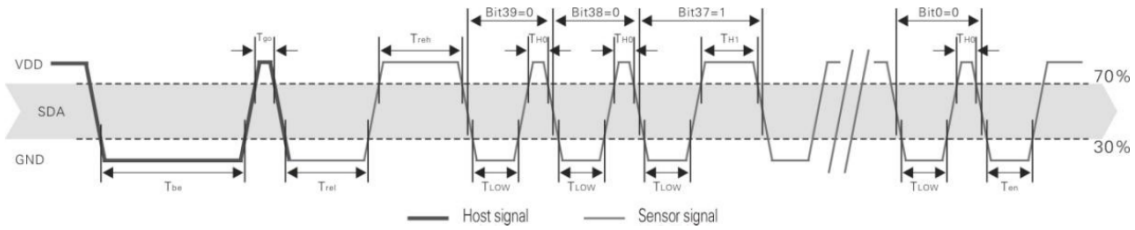

FIGURE 2.2 – Output de la température et de l'humidité sur l'ecran LCD

# Chapitre 3

## 3.1 Capteur DHT22

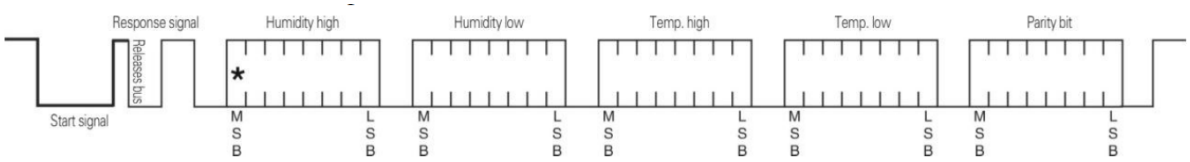
### 3.1.1 Protocole One Wire

Le protocole One Wire utilise un bus série asynchrone utilisant la largeur d'impulsion pour représenter les symboles. Comme son nom l'indique, ce protocole n'utilise qu'un fil et le bus ne véhicule pas de signaux de synchronisation (pas de signal d'horloge).
Les timings pour la communication One Wire peuvent varier d'un capteur à un autre.
Voici ci-dessous les timings correspondant au capteur DHT22 que nous avons mis en œuvre.




On peut noter qu'au repos, la ligne est à l'état haut.
La communication se passe de la façon suivante :
Pour initier la communication, le maître met le signal au niveau bas pendant une durée de 0.8 à 20ms (typ1 ms) puis le relâche pendant une durée de 20 à 200 µs (typ 30 µs). Le capteur répond en maintenant le signal à l'état bas entre 75 et 85 µs (typ 80 µs) puis le relâche à son tour pendant environ 80 µs. Ensuite, le capteur commence à envoyer les données.
Le code d'un bit est le suivant : le signal est mis à l'état bas pendant une durée de 48 à 55 µs (typ 50 µs) puis le signal repasse à l'état haut. C'est la durée de cet état haut qui

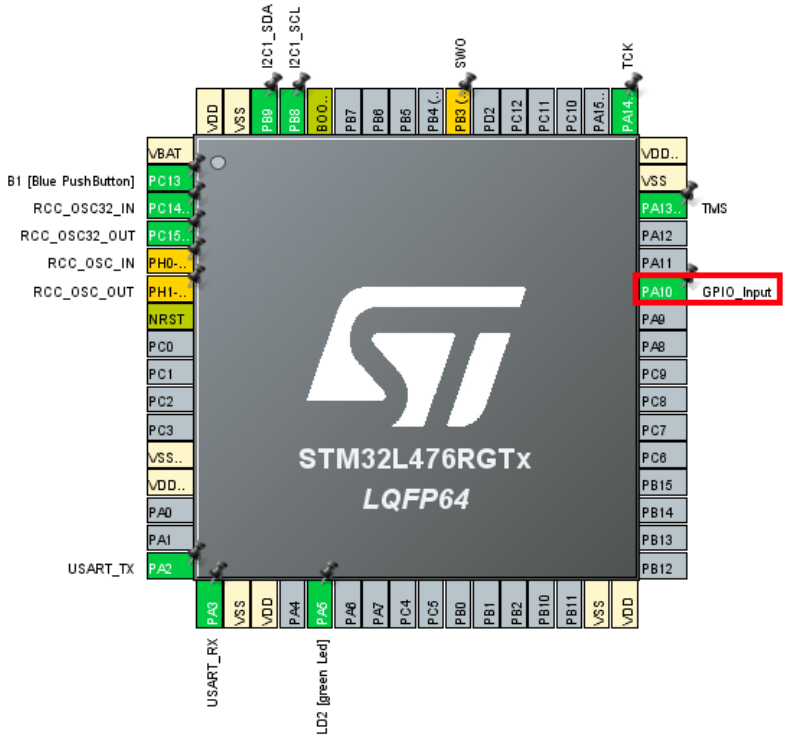
permet de distinguer un '0' d'un '1' : de 22 à 30 µs (typ 26 µs) pour le premier et de 68 à 75 (typ 70) pour le second. Ainsi, l'envoi d'un '1' dure plus longtemps que l'envoi d'un '0'. Le protocole de communication observé est le suivant :




Le microcontrôleur initie la communication, le capteur répond puis envoie les données dans l'ordre suivant : les 8 bits de poids forts de l'humidité puis ceux de poids faible, les 8 bits de poids forts de la température puis ici aussi ceux de poids faible et enfin 8 bits de parité pour contrôler les erreurs.

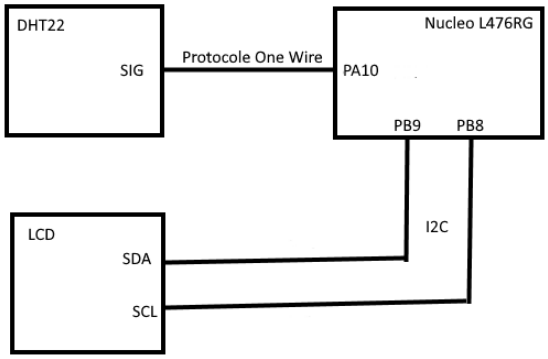
### 3.1.2 Schéma de cablâge

Pour relier le capteur et le microcontrôleur, nous avons utilisé la PIN 10 du GPIOA comme ci-dessous :



Il importe peu de paramétrer la PIN en sortie ou en entrée puis la communication One Wire requiert un partage du fil. La PIN sera donc à certains moments en entrée et à d'autre en sortie.

Le schéma de câblage correspondant est le suivant :




### 3.1.3 Programmation

#### Timer

Pour pouvoir utiliser le capteur, nous avons dû créer une fonction afin de générer un délai en µs.
Pour cela, nous avons modifié les paramètres du Timer 1 dans CUBE-MX comme ceci :
Nous avons ensuite créé la fonction *void delay_us (uint16_t us)*.

```
void delay_us (uint16_t us)
{
    __HAL_TIM_SET_COUNTER(&htim1,0);  // set the counter value a 0
    while (__HAL_TIM_GET_COUNTER(&htim1) < us);  // wait for the counter to reach the us input in the parameter
}
```

### Utilisation du capteur

Pour utiliser le capteur, nous avons créé plusieurs fonctions.
Tout d'abord, nous avons commencé par 2 fonctions *void setPinOutPut0(unsigned char x)* et *void setPinInput1(unsigned char x)* permettant respectivement de configurer la PIN PA10 en sortie en mettant le signal au niveau bas et en entrée en mettant le signal au niveau haut.
Afin de configurater des PINs, nous devons manipuler différents registres. Nous avons utilisé le tableau suivant pour nous aider.

**Table de configuration des broches d'un port**

| MODER(i) [1:0] | OTYPER(i) | OSPEEDR(i) [B:A] | | PUPDR(i) [1:0] | | I/O configuration | |
|---|---|---|---|---|---|---|---|
| 00 | x | x | x | 0 | 0 | Input | Floating |
| | x | x | x | 0 | 1 | Input | Pull-Up |
| | x | x | x | 1 | 0 | Input | Pull-Down |
| | x | x | x | 1 | 1 | Reserved (input floating) | |
| 01 | 0 | SPEED [B:A] | | 0 | 0 | GP output | Push-Pull |
| | 0 | | | 0 | 1 | GP output | Push-Pull+PU |
| | 0 | | | 1 | 0 | GP output | Push-Pull+PD |
| | 0 | | | 1 | 1 | Reserved | |
| | 1 | | | 0 | 0 | GP output | Open Drain |
| | 1 | | | 0 | 1 | GP output | Open Drain+PU |
| | 1 | | | 1 | 0 | GP output | Open Drain+PD |
| | 1 | | | 1 | 1 | Reserved (GP output OD) | |
| 10 | 0 | SPEED [B:A] | | 0 | 0 | Alternate Fu. | Push-Pull |
| | 0 | | | 0 | 1 | Alternate Fu. | Push-Pull+PU |
| | 0 | | | 1 | 0 | Alternate Fu. | Push-Pull+PD |
| | 0 | | | 1 | 1 | Reserved | |
| | 1 | | | 0 | 0 | Alternate Fu. | Open Drain |
| | 1 | | | 0 | 1 | Alternate Fu. | Open Drain+PU |
| | 1 | | | 1 | 0 | Alternate Fu. | Open Drain+PD |
| | 1 | | | 1 | 1 | Reserved | |
| 11 | x | x | x | 0 | 0 | Input/output | Analog |
| | x | x | x | 0 | 1 | Reserved | |
| | x | x | x | 1 | 0 | | |
| | x | x | x | 1 | 1 | | |

Nous avons ensuite utiliser ces fonctions pour créer la fonction ***void start_one_wire(unsigned char pin)***. Cette fonction permet d'initier la communication.
Après l'appel de cette fonction, nous pouvons utiliser les fonctions pour lire les données ***void read_one_wire(uint8_t * data)*** **et les convertir** ***void conversion(uint8_t * data, float * temp_partE, float * temp_partD, float * hum_partE, float * hum_partD).***
Nous pouvons ensuite afficher les données sur le LCD

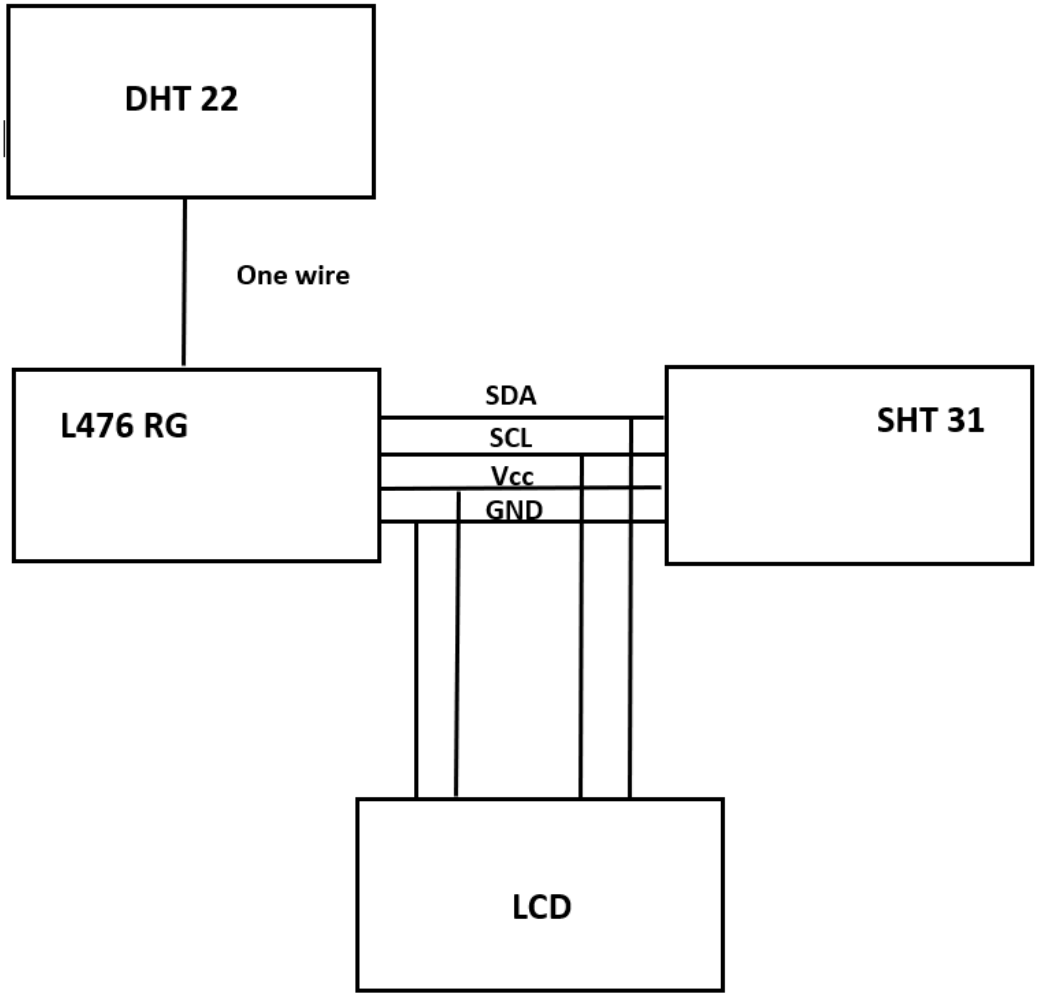
### 3.1.4 Trame observée

Nous avons relevé la trame observée sur un oscilloscope.
Nous avons mis en rouge la partie de la trame générée grâce à la fonction *void start_one_wire(unsigned char pin)*.

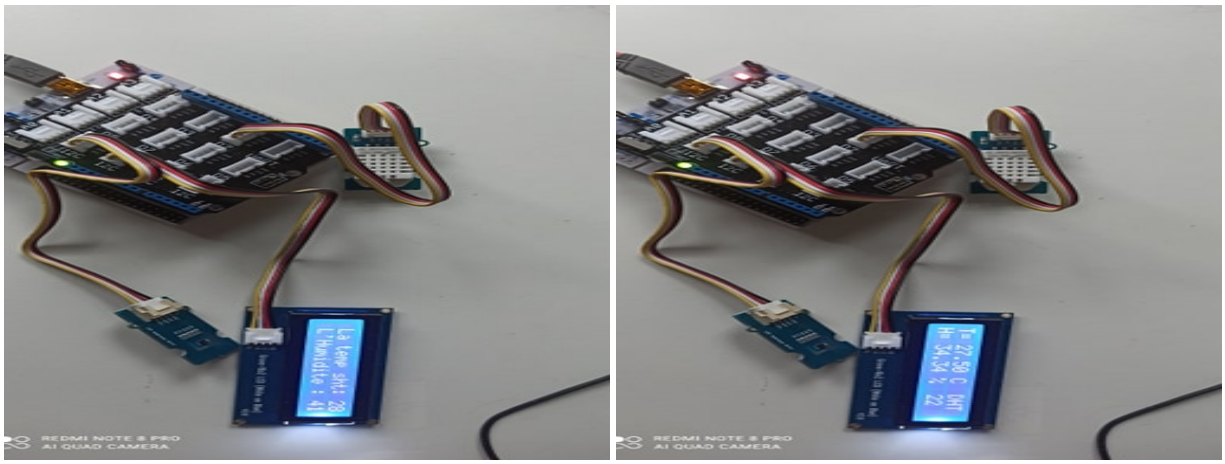
# Chapitre 4

## 4.1 SHT31 + DHT22 :

Dans cette partie nous allons regrouper les deux capteurs suivant le schéma de cablage ci dessous :



Nous avons obtenu le resultat suivant :



# Conclusion

Ces TP de base ont été l'occasion de manipuler une carte Nucleo L476RG.
Nous avons utilisé deux capteurs utilisant deux protocoles différents (One Wire et I2C) ainsi qu'un écran LCD. Nous avons également pu nous initier à l'utilisation de CUBE-MX ainsi que de l'IDE TrueStudio.