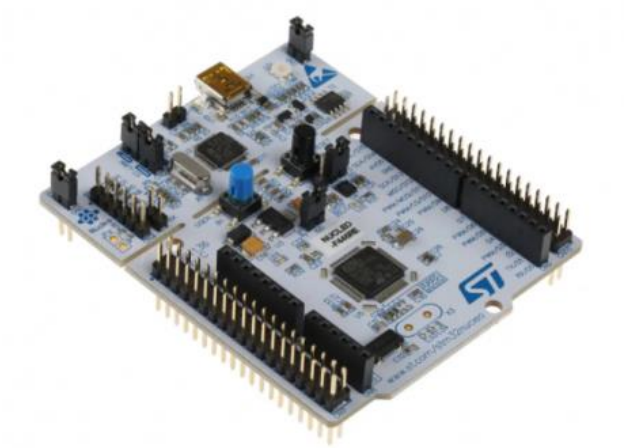
UNIVERSITE TOULOUSE III PAUL SABATIER
Faculté Sciences et Ingénierie

Compte rendu - Réalisation système



TP de base STM32L152RE avec LCD I2C et DHT22 one wire

JACQ Léo

Année 2021/2022

Table des matières

Matériel

Le matériel utilisé dans le cadre de ce TP sera une carte SMT32 NUCLEO L152RE, un capteur humidité/température DHT22 et un afficheur LCD.

Pour faire une brève description des caractéristiques des composants nous avons :

Plateforme ARM 32 bits basse consommation de 32Mhz et 64 broches.

Capteur d'humidité/température communication one wire.

LCD 2lignes 16 colonnes RGB communication I2C.

Le tout est connecté grâce à un base shield V2 pour capteur grove. Ainsi la partie câblage est donc simplifiée.

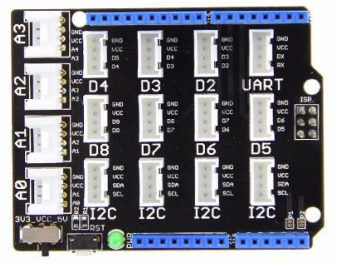


Figure 1 : Base shield arduino

Toutefois cette partie n'a pas totalement négligée car nécessaire au bon paramétrage du microcontrôleur.

Schéma et câblage

Le brochage des composants se fera comme suit :

Broche MCU	Broche capteur
PA1	Signal (SGN)
PB8	SCL
PB9	SDA

Tableau 1 : Brochage

La communication I2C et la gestion des entrées/sorties sont directement gérées par les différentes bibliothèques disponibles. La communication one-wire a été faite manuellement en modifiant la direction de la broche PA1 pour alterner entre entrée/sortie.

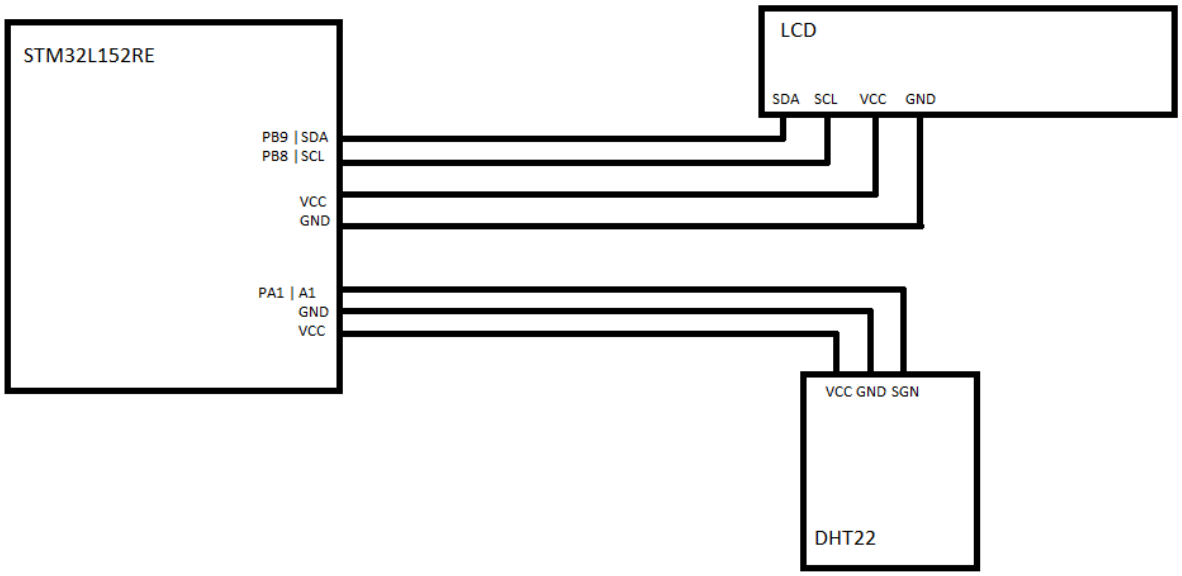


Figure 2 : Base câblage des composants

Programme

Principe

Le diagramme ci-dessous présente simplement la routine du programme permettant la collecte des données de températures et d'humidité ainsi que l'affiche sur l'écran LCD.

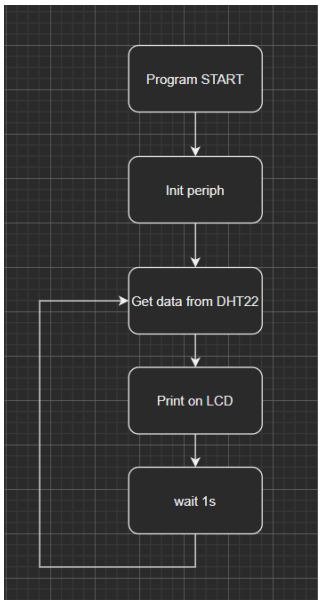


Figure 3 : Principe du programme

Projet

La mise en place des fichiers de projet se fait au démarrage grâce à l'outil STM32CubeMX qui est un outil graphique de configuration de microcontrôleur. Il permet de générer un code de base pour initialiser simplement les périphériques et IO du microcontrôleur.

Ce logiciel permet donc de programmer les IO mais aussi de gérer les différentes clocks.

En plus des deux bus de communications pour le LCD et le capteur nous avons eu besoin d'un timer externe de 32 Mhz avec un pré-diviseur de 32 afin d'obtenir des périodes de 1microsecondes, unité de temps nécessaire à l'exploitation du DHT22.

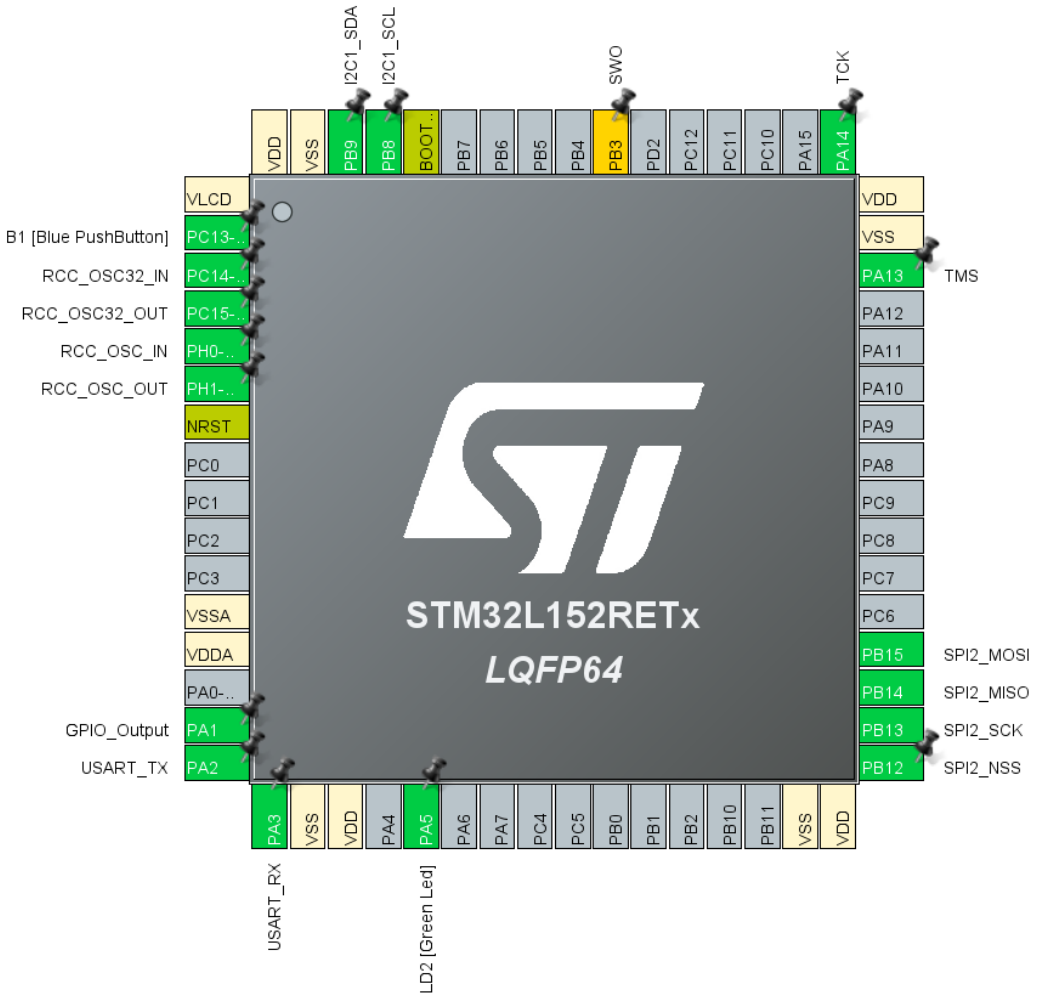


Figure 4 : Pin affectés

Délais microseconde

Initialisation

La couche d'abstraction matérielle ne comporte pas de fonction délais prenant en charge les microsecondes. Pour ce faire il nous faut donc dédier un timer afin d'obtenir les fonctions souhaitées. J'ai donc paramétré le timer 2 avec une fréquence de 32mhz et un prédiviseur de 32 afin d'obtenir une fréquence de fonctionnement de 1Mhz soit une microseconde de période.

La fonction delay_us() va donc prendre en paramètre un entier correspondant à la période d'attente. Au départ, le compteur du timer va être remis à 0 et la boucle while va s'exécuter tant que le compteur n'a pas atteint la valeur spécifié dans la fonction

```
/* USER CODE BEGIN 1 */
void delay_us(uint16_t us)
{
    __HAL_TIM_SET_COUNTER(&htim2,0);  // set the counter value a 0
    while (__HAL_TIM_GET_COUNTER(&htim2) < us); // wait for the counter to reach the us input in the parameter
}

/* USER CODE END 1 */
```

Figure 5 : Fonction délais microseconde

Code DHT22

Fonctionnement du capteur

Avant de tenter de récupérer les données, il m'a fallu comprendre comment fonctionnait le capteur et donc lire la documentation technique de ce dernier.

Comme énoncé plus haut le capteur fonctionne en one wire, c'est donc la même broche qui servira d'entrée pour les demandes de conversion et de sortie pour l'envoi des données. Le protocole de communication est défini comme suit :

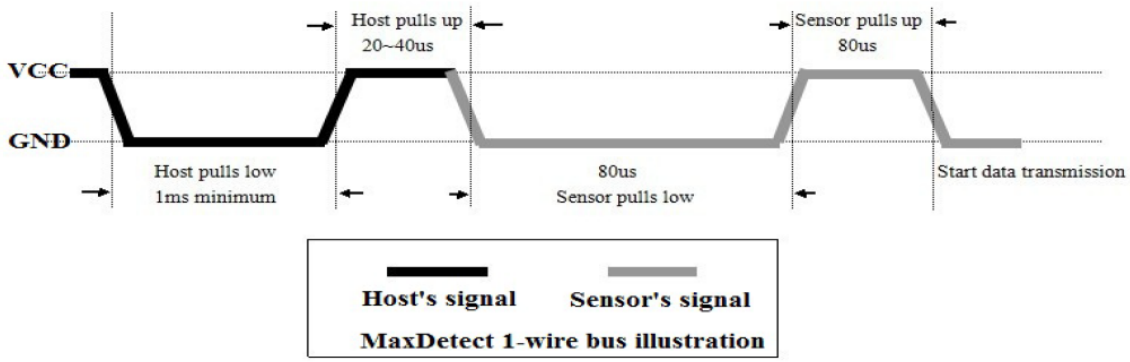


Figure 6 : Séquence de réveil du capteur

Au repos, la ligne est à l'état haut, pour se « réveiller » le capteur attend un signal à l'état bas pendant au moins 1ms, puis un signal haut entre 20 et 40us. A ce moment-là le capteur va répondre en mettant le signal l'état bas pendant 80us puis à l'état haut pendant 80us.

Une fois cette séquence écoulée commence l'envoi de la trame de données de 40 bits.

Chaque bit est séparé d'un signal de 50us à l'état bas puis le temps à l'état haut définira si le bit est un 1 ou un 0.

Le 0 est défini par un signal haut entre 26 et 28us.

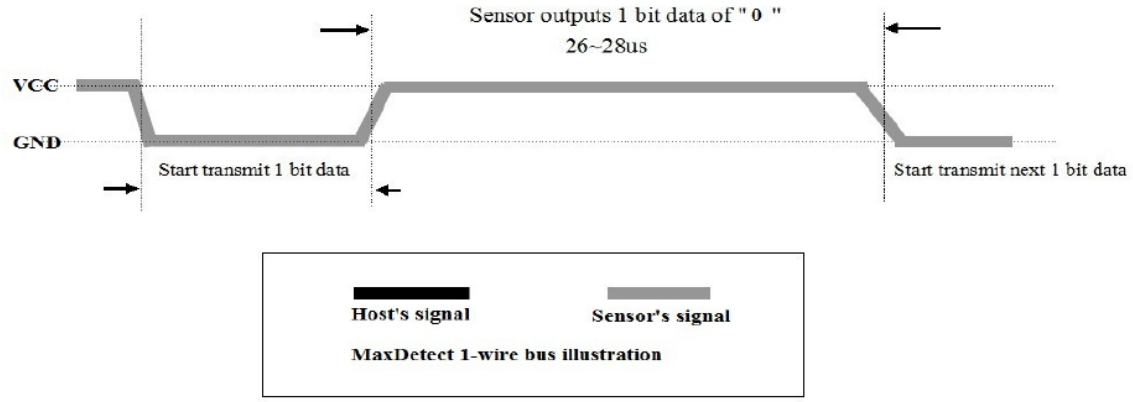


Figure 7 : Envoi 0

Le 1 est défini par un signal haut de 70us.

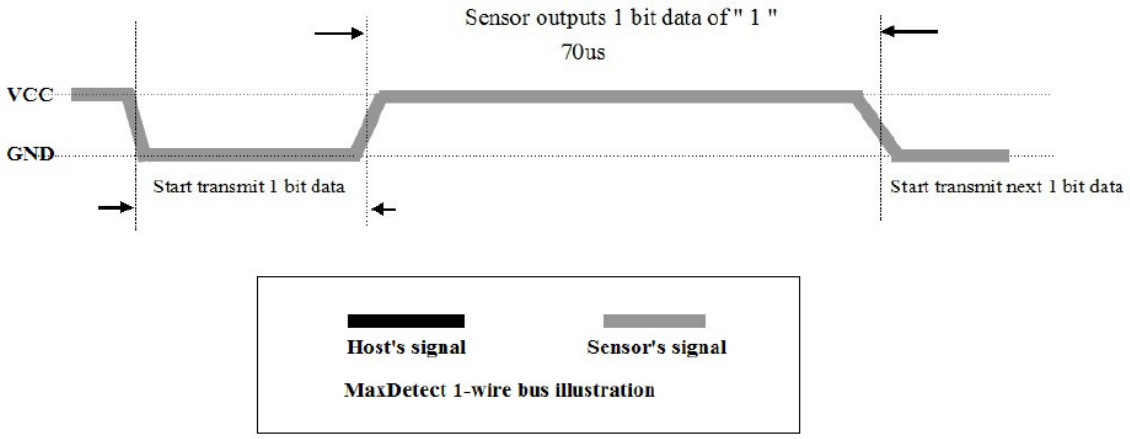


Figure 8 : Envoi 1

Une fois la séquence de 40 bits reçue il nous faut la décoder.

Octet 1	Octet 2	Octet 3	Octet 4	Octet 5
Humidité		Température		Checksum

Tableau 2 : Trame DHT22

Les données brutes d'humidité et de température sont codées sur 16 bits. Pour obtenir les valeurs de mesurandes il faut convertir ces 16 bits en base décimale et diviser le tout par 10.

Les 8 derniers bits servent au checksum pour la sécurité. Son calcul se fait par addition de tous les octets de données de la trame (octets 1-4). Si la valeur obtenue est identique à celle de la trame alors il n'y a pas d'erreur de transmissions, dans le cas contraire la trame a été altérée.

Initialisation

Plusieurs tutoriels et exemples d'utilisations à propos du DHT22 existent mais soit ne fonctionnait pas soit laissait des failles dans l'utilisation du programme.

J'ai donc développé ma propre bibliothèque afin d'utiliser le module plus simplement.

La fonction d'initialisation prend en paramètres une structure de type DHT22 ainsi que les ports et pin physiques reliés au module.

```
struct DHT22{

    GPIO_TypeDef *GPIOx;
    uint16_t GPIO_Pin;
    float temperature;
    float humidity;
};
```

Figure 9 : Structure DHT22

La structure DHT22 permet de définir un objet à chaque module en gardant en paramètre les données de températures et humidité mais aussi de raccordement au microcontrôleur, ainsi on peut utiliser facilement et distinctement plusieurs capteurs dans un seul programme.

Mesure

Comme nous avons pu le voir la récupération des données s'effectue en trois étapes :

-Réveil du capteur

-Réponse du capteur

-Envoi des données

```
if(DHT22_Start(&DHT22_1) == 0)
{
    if( DHT22_Check_Response(&DHT22_1)== 0)
    {
        if(DHT22_Read_Temp_Hum(&DHT22_1) == 0)
        {
            clearlcd();

            lcd_position(&hi2c1,0,0);

            memset(temp,0,sizeof(temp));
            sprintf((char*)temp,"Hum: %.2f %c ",DHT22_1.humidity, 0x25);
            lcd_print(&hi2c1,(char*)temp);

            lcd_position(&hi2c1,0,1);

            memset(temp,0,sizeof(temp));
            sprintf((char*)temp,"Temp: %.2fC  ",DHT22_1.temperature);
            lcd_print(&hi2c1,(char*)temp);

            HAL_Delay(1000);
        }
        else
        {
            HAL_Delay(10);
        }

    }
    else
    {
        HAL_Delay(10);
    }

}
else
{
    HAL_Delay(10);
}
```

Figure 10 : Corps du while

La fonction DHT22_Start() prend en paramètre l'adresse de la structure DHT22. Et joue la séquence de réveil spécifié sur la Figure 6.

```
/*Set pinmode to output and send > 1ms low signal,  20-40 us high signal and set input*/
uint8_t DHT22_Start (struct DHT22 *sensor_DHT22)
{
    DHT22_Set_Output(sensor_DHT22);
    HAL_GPIO_WritePin (sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin, GPIO_PIN_RESET);
    delay_us(1200);
    HAL_GPIO_WritePin (sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin, GPIO_PIN_SET);
    delay_us(30);
    DHT22_Set_Input(sensor_DHT22);

    return 0;
}
```

Figure 11 : Fonction DHT22_Start()

Une fois la séquence effectuée le pin est définie en entrée grâce à la fonction DHT22_Set_Input(), cette fonction va redéfinir les paramètres générés par CubeMX afin de définir le pin du DHT222 comme une entrée.

```
void DHT22_Set_Input (struct DHT22 *sensor_DHT22)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = sensor_DHT22->GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(sensor_DHT22->GPIOx, &GPIO_InitStruct);
}
```

Figure 12 : Fonction DHT22_Set_Input()

La même fonction pour rebasculer le pin en sortie a été implémentée.

```
void DHT22_Set_Output(struct DHT22 *sensor_DHT22)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = sensor_DHT22->GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(sensor_DHT22->GPIOx, &GPIO_InitStruct);

}
```

Figure 13 : Fonction DHT22_Set_Output()

La prochaine étape est donc de regarder la réponse du capteur suite à son activation. Pour ce faire et comme mentionné dans la datasheet du composant nous devant attendre deux changements d'états espacés chacun de 80 microsecondes. Afin d'éviter un blocage du programme dans une boucle ou un cas où le capteur ne répondrait pas ou serait débranché, un contrôle de sécurité est effectué pour terminer l'attente. Dans ce cas, est renvoyé une erreur pour la fonction afin d'indiquer que la communication a échouée.

```
/*Wait sensor response, 80 us low signal and 80 us high signal*/
uint8_t DHT22_Check_Response (struct DHT22 *sensor_DHT22)
{
    uint8_t wd_timer = 0;
    while(!(HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 85))
    {
        delay_us(1);
        wd_timer++;
    }

    if(wd_timer == 85)
    {
        return 1;
    }
    else
    {

        wd_timer = 0;
        while((HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 85))
        {
            delay_us(1);
            wd_timer++;
        }

        if(wd_timer == 85)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }

}
```

Figure 14 : Fonction DHT22_Check_Response()

Une fois le capteur réveillé et ayant répondu, ce dernier va envoyer les données sur 40 bits

La fonction DHT22_Read_raw va lire un octet de données bit pas bit en calculant le temps à l'état haut du signal du capteur et ainsi déterminer s'il s'agit d'un 0 ou d'un 1. L'octet est renvoyé dans un pointeur ce qui permet de gérer les erreurs si le signal ne correspond pas avec les données attendues.

```c
uint8_t DHT22_Read_raw (struct DHT22 *sensor_DHT22, uint8_t * data)
{
    uint8_t i = 0;
    uint8_t wd_timer = 0;

    for (i=0;i<8;i++)
    {
        wd_timer = 0;

        //Start bit of 50us
        while(!(HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 50))
        {
            delay_us(1);
            wd_timer++;
        }

        wd_timer = 0;

        while((HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 90))
        {
            wd_timer += 10;
            delay_us(10);
        }


        if((wd_timer >= 20)&&(wd_timer <= 30 ))
        {
            *data &= ~1<<(7-i);
        }
        else if((wd_timer >= 60)&&(wd_timer <= 80 ))
        {
            *data |= 1<<(7-i);
        }
        else
        {
            return 1;
        }
    }

    return 0;
}
```

Figure 15 : Fonction DHT22_Read_Raw()

Chaque octet de la trame va donc pouvoir être lu à la suite et utilisé pour la calcul de la checksum. Si le calcul est juste alors on peut actualiser les données de l'objet DHT22, dans le cas connaitre on renvois une erreur.

Pour obtenir les valeurs de température et d'humidité il ne reste plus qu'a assembler les octets de poids forts et octets de poids faible de chaque mesure puis de diviser par 10 comme spécifié dans le datasheet et les données peuvent maintient être affichées.

```c
uint8_t DHT22_Read_Temp_Hum (struct DHT22 *sensor_DHT22)
{
    uint8_t Rh_byte1 = 0, Rh_byte2 = 0, Temp_byte1 = 0, Temp_byte2 = 0 , SUM = 0, SUM_temp = 0, read_error = 0;
    uint16_t RH = 0, TEMP = 0;


      read_error += DHT22_Read_raw(sensor_DHT22, &Rh_byte1);
      read_error += DHT22_Read_raw(sensor_DHT22, &Rh_byte2);
      read_error += DHT22_Read_raw(sensor_DHT22, &Temp_byte1);
      read_error += DHT22_Read_raw(sensor_DHT22, &Temp_byte2);
      read_error += DHT22_Read_raw(sensor_DHT22, &SUM);

      if(read_error == 0)
      {
          SUM_temp = Rh_byte1 + Rh_byte2 + Temp_byte1 + Temp_byte2 ;

          if(SUM == SUM_temp)
          {

              TEMP = ((Temp_byte1<<8)|Temp_byte2);
              RH = ((Rh_byte1<<8)|Rh_byte2);

              sensor_DHT22->temperature = (float) (TEMP/10.0);
              sensor_DHT22->humidity = (float) (RH/10.0);
              return 0;
          }
          else
          {
              return 1;
          }
      }
      else
      {
          return 1;
      }
}
```

Figure 16 : Fonction DHT22_Read_Temp_hum()

Code LCD

Initialisation

Les fonctions de gestion du LCD ont été prises dans une bibliothèque disponible en ligne et développée par Loovee de Seeed technology Inc.

Cette bibliothèque est assez clé en mains et nous permet d'utiliser le LCD simplement grâce à des fonctions prédéfinies sans avoir à reconstruire les différentes trames en fonction des actions que l'on veut effectuer.

Pour initialiser le module nous avons besoin de deux structures, la première du type I2C_HandleTypeDef et la seconde du type rgb_lcd. Respectivement ces structures comportent les paramètres physiques de la ligne I2C ainsi que ceux de l'objet LCD. Ces deux paramètres sont passés à la fonction lcd_init.

Affichage

Les trois fonctions qui nous intéressent dans notre cas d'utilisation sont clearlcd(), lcd_print() et lcd_position(). Lcd_position() permet de préciser la place du curseur en passant en paramètre la position de la colonne puis de la ligne.

La partie affichage de température et Humidité ressemble donc à cela :

```
clearlcd();

lcd_position(&hi2c1,0,0);

memset(temp,0,sizeof(temp));
sprintf((char*)temp,"Hum: %.2f %c ",DHT22_1.humidity, 0x25);
lcd_print(&hi2c1,(char*)temp);

lcd_position(&hi2c1,0,1);

memset(temp,0,sizeof(temp));
sprintf((char*)temp,"Temp: %.2fC  ",DHT22_1.temperature);
lcd_print(&hi2c1,(char*)temp);

HAL_Delay(1000);
```

Figure 17 : Affichage LCD

Avant chaque affichage le LCD est effacé puis le reste s'effectue en trois temps :

Affichage de l'humidité, repositionnement à la deuxième ligne puis affichage de la température.

L'affichage des deux variables est similaire :

- La fonction memset() permet la mise à 0 du vecteur temp (uint8_t de longueur 16).
- Construction de la donnée avec la fonction sprintf() qui formate une chaîne dans une variable de sortie avec les paramètres souhaités (similaire à l'utilisation de printf)
- Lcd_print envoie une commande d'écriture sur le LCD avec la chaîne souhaitée.

Le Hal_Delay est une fonction HAL de base permettant d'obtenir un délai en milliseconde. Ici de 100 afin de fixer la fréquence de prise de données et affichage a 1 Hz.

Conclusion

Ces TP de base nous ont permis de prendre en mains l'interface CubeMx ainsi que différents capteurs avec des protocoles de communications plus ou moins complexes. Ce qui va nous permettre de mieux appréhender la réalisation du projet de BE.